郑州大学毕业设计（论文）

题 目： 基于MI1000的波形和时序生成工具

指导教师： 郑志蕴 职称： 教授

学生姓名： 郭攀 学号： 20102480211

专 业： 软件工程

院（系）： 信息工程学院

完成时间： 2014年5月4日

2013年5月4日

基于MI1000的波形和时序生成工具

**摘要** 项目《基于MI1000的波形和时序生成工具》是为了开发一个给不同知识背景用户使用并生成他们想要的波形数据或时序数据的软件，它通过对生成波形或时序数据的图形化展示使得用户确认是否是它们想要的波形；当用户获取到他们想要的数据后，可以通过MI1000系列硬件生成与数据对应的电气信号，用来满足自己的需要。

项目中生成的波形或时序数据是基于MI1000系列硬件的，因此它需要符合MI1000定义的数据规范；项目中采用线性插值算法，用以在同时考虑高效与高保真的情况下图形化显示波形或时序数据；项目使用MVC框架模式，使得项目分工更加明确，项目的模块重用性也更强；项目最终开发使用的是Qt图形用户界面应用程序框架，使得开发人员开发软件更为简单，可以花费更多的时间在程序实现逻辑上而不是细节上。

**关键词** MI1000、预览、波形生成、时序生成

目录

[基于MI1000的波形和时序生成工具 2](#_Toc389055097)

[1 背景 5](#_Toc389055098)

[1.1 项目介绍 5](#_Toc389055099)

[1.2 国内外研究现状 5](#_Toc389055100)

[2 系统相关技术和工具 6](#_Toc389055101)

[2.1 MI1000系列硬件 6](#_Toc389055102)

[2.2 线性插值算法 6](#_Toc389055103)

[2.3 MVC框架模式 7](#_Toc389055104)

[2.3.1 MVC框架模式规定的操作流程 7](#_Toc389055105)

[2.3.2 MVC框架模式的特性 8](#_Toc389055106)

[2.3.3 使用MVC框架模式开发应用程序的优势 8](#_Toc389055107)

[2.4 图形用户界面应用程序框架Qt 8](#_Toc389055108)

[3 系统分析与设计 9](#_Toc389055109)

[3.1 需求分析 9](#_Toc389055110)

[3.1.1 功能性需求 9](#_Toc389055111)

[3.1.2 非功能性需求 10](#_Toc389055112)

[3.1.3 外部接口 10](#_Toc389055113)

[3.1.4 时间要求 10](#_Toc389055114)

[3.2 总体设计 11](#_Toc389055115)

[3.2.1 设计思路 11](#_Toc389055116)

[3.2.2 总体框架 11](#_Toc389055117)

[3.2.3 时间进度安排 12](#_Toc389055118)

[3.3 详细设计 12](#_Toc389055119)

[3.3.1 操作流程 12](#_Toc389055120)

[3.3.2 模块划分 14](#_Toc389055121)

[3.3.3 选择协议或波形模块设计 14](#_Toc389055122)

[3.3.4 预览模块设计 15](#_Toc389055123)

[3.3.5 绘图板模块设计 17](#_Toc389055124)

[3.3.6 内部接口设计 17](#_Toc389055125)

[4 系统具体实现 17](#_Toc389055126)

[4.1 内部接口的实现 17](#_Toc389055127)

[4.2 选择协议或波形模块的实现 18](#_Toc389055128)

[4.3 模拟信号预览模块的实现 21](#_Toc389055129)

[4.4 数字信号预览模块的实现 26](#_Toc389055130)

[4.5 绘图板模块的实现 33](#_Toc389055131)

[4.6 程序主体框架的实现 35](#_Toc389055132)

[5 总结 38](#_Toc389055133)

# 背景

科学技术的发展推动了社会的发展与变化，方便并丰富了人们的生活。现代通信技术的进步，使得古人幻想的千里眼和顺风耳变成了现实。藉此，我们可以很快了解到自己周围、祖国各地、世界各国发生的重要事情，使得地球真正的变成了一个地球村。

现代通信技术的核心是通信协议。通过通信信道和设备互连起来的多个不同地理位置的数据通信系统，要使其能协同工作实现信息交换和资源共享，它们之间必须具有共同的语言——通信协议。

通信协议（communications protocol）是指双方实体完成通信或服务所必须遵循的规则和约定。协议定义了数据单元使用的格式，信息单元应该包含的信息与含义，连接方式，信息发送和接收的时序，从而确保网络中数据顺利地传送到确定的地方。

## 项目介绍

软件是基于MI1000系列硬件的配套工具。目的是方便用户产生相应的波形和时序数据，为逻辑分析仪、示波器插件分析模块提供波形和时序支持；同时将它保存成文件，可以由Mistudio（MI1000系列硬件配套软件）配合MI1000硬件产生用户需要的电气信号。

## 国内外研究现状

信号发生器是一种常用的信号源，广泛用于电子电路、自动控制和科学实验等领域。它是一种为电子测量和计量工作提供符合严格技术要求的电信号设备。

自六十年代以来，信号发生器有了迅速的发展，出现了函数发生器、扫频信号发生器、合成信号发生器、程控信号发生器等新种类。各类信号发生器的主要性能指标也都有了大幅度的提高，同时在简化机械结构、小型化、多功能等各方面也有了显著的进展。

近年来随着GSM、GPRS、3G、BlueTooth乃至已经被提出的标准的4G等移动通信以及LMDS、无线本地环路等无线接入的发展，同时加上合成孔径雷达、多普勒冲雷达等现代军事、国防、航空航天等在科技上的不断创新与进步，世界各国非常重视频率合成器的发展。所有的这些社会需求以及微电子技术、计算机技术、信号处理技术等本身的不断进步都极大刺激了频率合成技术的发展。可以预料，随着低价格、高时钟频率、高性能的新一代DDS芯片的问世，DDS的应用前景将不可估量。

# 系统相关技术和工具

## MI1000系列硬件

MI1000系列硬件是致远电子推出的多功能虚拟仪器。它揉合了日常电路调试过程中所需的信号产生、测量等多种功能，构建了一个功能强大的闭环测试系统，既有数字、模拟信号的激励，亦有数字、模拟信号的检测，且逻辑分析仪还嵌入了常用信号的协议分析功能。

六种功能有机地结合在一起，功能模块间可协同工作，控制简单，观测方便，在有效地节约工作台面积的同时，也为信号调试提供了极具性价比的仪器解决方案。作为一款虚拟仪器，其充分利用了上位机强大的数据处理能力，为用户预留二次开发的程序接口。

它集六种仪器于一身：

* 数字示波器：双通道，60MHz带宽，4M最大储存深度。
* 逻辑分析仪：32通道输入。
* 信号发生器：单通道，最高40MHz正弦波输出，支持多种信号调制。
* 码型发生器：16通道输出，多种跳转指令，用于数字信号激励。
* 协议分析：支持A/D总线、UART、I2C、SPI等。
* 扫频仪：测量器件或设备传递函数的幅频特性。

## 线性插值算法

线性插值是数学、计算机图形学等领域广泛使用的一种简单插值方法。它经常用来补充函数表中不存在的表项。以下是对线性插值算法的介绍：

假设我们已知坐标(x0,y0)与(x1,y1),要得到[x0,x1]区间内某一位置x在(x0,y0)与(x1,y1)构成的直线上的y值。

首先得到（y-y0）(x1-x0)=(y1-y0)(x-x0)。

然后假设方程两边的值为a，那么这个值就是插值系数—从x0到x的距离与从x0到x1距离的比值。由于x值已知，所以可以从公式得到a的值：a=(x-x0)/(x1-x0)。同样，a=(y-y0)/(y1-y0)

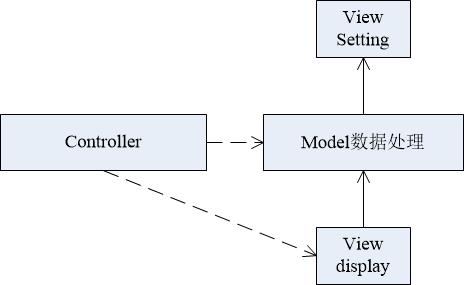
最后，在代数上就可以表示成为：y = y0 + a(y1 - y0)。

## MVC框架模式

MVC是一个框架模式，它强制性的使应用程序的输入、处理和输出分开。使用MVC应用程序被分成三个核心部件：模型（Model）、视图（View）、控制器（Ccontrol）。它们各自处理自己的任务。MVC结构提供了一种按功能对各种对象进行分割的方法（这些对象是用来维护和表现数据的），其目的是为了将各对象间的耦合程度减至最小。MVC被广泛应用于网站设计，但是实际上它适用于任何类型的需要处理较为复杂数据并显示相关信息的应用程序。

### MVC框架模式规定的操作流程

图片 2‑1为MVC框架一般结构



图片 2‑1 MVC框架一般结构

1. 用户发出请求
2. Controller获取请求，并通知Model数据处理模块向ViewSetting请求数据
3. ViewSetting收集当前显示层的数据，并交付Model数据处理模块
4. Model数据处理模块获取数据，生成相应的数据
5. Controller通知ViewDisplay从Model数据处理请求数据
6. Model数据处理模块将波形数据交付给ViewDisplay
7. ViewDisplay进行显示

### MVC框架模式的特性

MVC框架模式有以下特性：

* 多个视图可以对应一个模型。按MVC设计模式，一个模型对应多个视图，可以减少代码的复制及代码的维护量，一旦模型发生改变，也易于维护。
* 应用被分隔为三层，降低了各层之间的耦合，提供了应用的可扩展性。
* MVC更符合软件工程化管理的精神。不同的层各司其职，每一层的组件具有相同的特征，有利于通过工程化和工具化产生管理程序代码。

### 使用MVC框架模式开发应用程序的优势

MVC 分层有助于管理复杂的应用程序，可以强制性地使得开发人员在一个时间内只关注一个方面，如设计试图的开发人员不需要考虑业务实现。同时它简化了分组开发，不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

## 图形用户界面应用程序框架Qt

Qt是1991年奇趣科技开发的一个跨平台的C++图形用户界面应用程序框架。它提供给应用程序开发者建立艺术级的图形用户界面所需的所有功能。Qt很容易扩展，并且允许真正地组件编程。基本上，Qt 同 X Window 上的 Motif，Openwin，GTK 等图形界面库和 Windows 平台上的 MFC，OWL，VCL，ATL 是同类型的东西。

总的来说，使用Qt开发应用程序具有以下优势：

1. 优良的跨平台特性

Qt支持下列操作系统:MS/Windows - 95、98、NT4.0、ME、2000、XP、Vista、Win7、win8、win2008、Unix/X11 - Linux、SunSolaris、HP-UX、CompaqTru64 UNIX、IBMAIX、SGI IRIX、FreeBSD、BSD/OS和其它很多X11平台、Macintosh -Mac OS X、Embedded - 有帧缓冲(framebuffer)支持的嵌入式Linux平台、Windows CE

1. 面向对象

Qt 的良好封装机制使得 Qt 的模块化程度非常高，可重用性较好，对于用户开发来说是非常 方便的。 Qt 提供了一种称为 signals/slots 的安全类型来替代 callback，这使得各个元件之间的协同工作变得十分简单。

1. 丰富的 API

Qt 包括多达250个以上的C++类，还提供基于模板的collections， serialization，file，I/O device，directory management，date/time 类。甚至还包括正则表达式的处理功能。

1. 支持 2D/3D 图形渲染，支持OpenGL
2. 大量的开发文档
3. XML 支持。

# 系统分析与设计

## 需求分析

### 功能性需求

软件是基于MI1000系列硬件配套工具。它的主要功能包括：

基本功能：数字信号包括UART、CAN、Wiegand协议的设置、生成、预览、保存功能；模拟信号包括正弦波、方波、脉冲波、梯形波、三角波、锯齿波的设置、生成、预览、保存功能。

扩展功能：SPI、1-wire、LIN、I2C协议数字信号的设置，生成，预览，保存功能；绘图板功能（可以由用户绘制自己想要的模拟波形）。

### 非功能性需求

必须满足的非功能性指标：

* 软件必须具有可扩展性：软件需要很方便的扩展可以处理的波形或时序的集合。
* 软件必须具有可移植性：软件有可能在将来移植到其他的系统平台上。
* 波形或时序预览时，绘制的图形不能出现锯齿。
* 软件至少可以在连续10个小时，并且伴随不定时操作的情况下正确处理用户的操作。
* 当用户设置波形或时序参数后，到要被绘制的图形出现之间的时间不能超过1s。
* 软件具有易用性，可以满足种知识背景的用户的使用。

### 外部接口

可保存生成的数据，数据格式为遵循下面原则。

* 模拟信号数据文件：数据文件为文本文件；一个模拟信号波形周期有2048个点，对模拟函数进行采样，将采样数据进行处理，最小值为0，最大值为1023。
* 数字信号数据文件：数据文件为二进制文件；共包括4096个数据，第0~2048个数据表示数据，第2049~4096数据表示命令；每个数据使用32位存储，每一位代表协议的一条总线。

### 时间要求

项目截日期为：2014年4月40日。

## 总体设计

### 设计思路

1. 系统定位

首先，需要明确系统的定位：

* 它是基于MI1000系列硬件的配套工具
* 它是为逻辑分析仪、示波器插件分析模块提供时序和波形支持

1. 核心问题

当明确系统的定位之后，抽取这个项目的核心问题：如何根据用户的需求生成特定格式的文件？

分析这个核心问题，可以将它按照以下特点来进行解析：

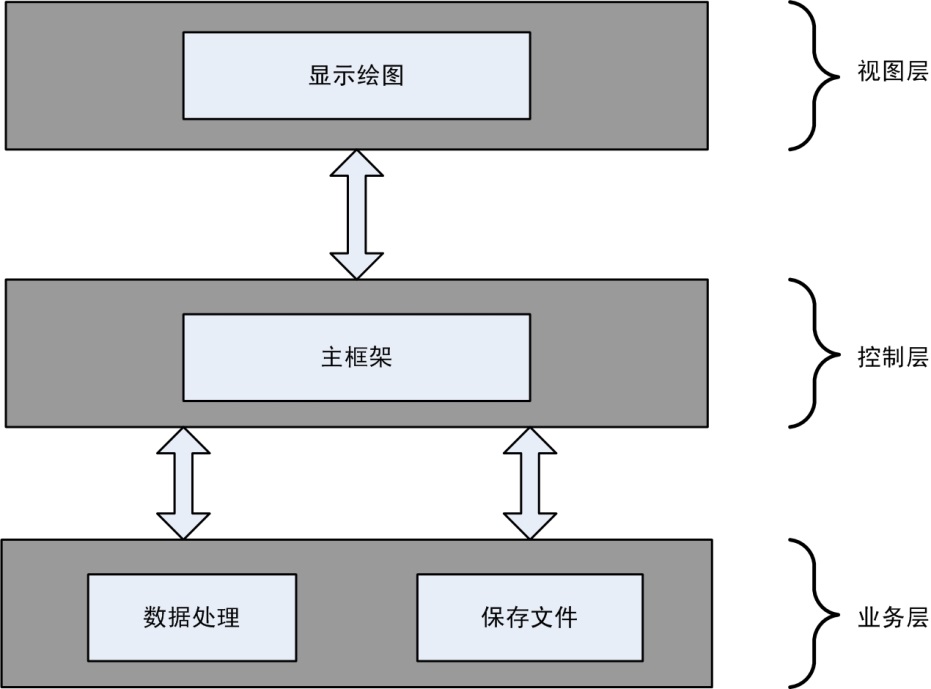
* 输入：用户的操作，也就是用户对波形或协议的设置
* 处理：根据用户输入生成数据
* 输出：根据生成的数据绘制相应的波形或时序、根据生成的数据保存成特定格式的文件

1. 细化处理流程

根据以上的处理流程以及实际情况，将处理流程进行细化为五个部分：选择协议或波形、协议或波形配置、数据生成、波形或协议预览、保存文件。

### 总体框架

为了降低耦合度，项目使用了MVC的架构，软件整体结构设计如图片 3‑1



图片 3‑1 总体框架图

### 时间进度安排

表格 3‑1为根据项目截至时间与实际情况制定的项目进度安排。

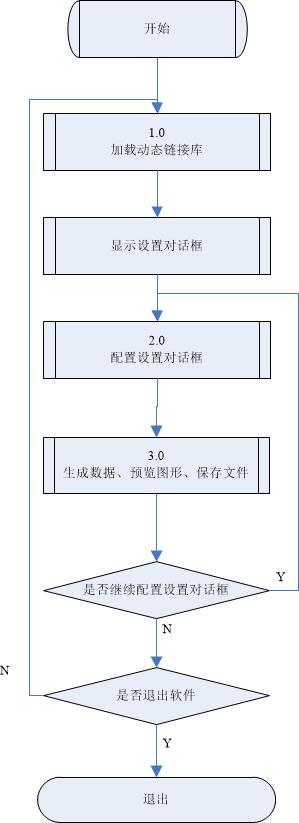
表格 3‑1 项目进度计划安排

|  |  |
| --- | --- |
| 2114.2.17至2014.2.23 | 收集所需资料。完成开题报告 |
| 2014.2.24至2014.3.8 | 完成系统需求分析，对各个功能模块进行确定 |
| 2014.3.9至2014.3.23 | 完成设计工作以及测试代码及实例编写工作 |
| 2014.3.24 至2014.4.20 | 完成功能模块代码编写 |
| 2014.4.20至2014.4.27 | 对系统进行测试，找出缺陷进行完善 |
| 2014.4.28至2014.5.5 | 按照公司规定编写论文并完成公司中的答辩 |
| 2014.5.7至2014.5.20 | 完成学校的毕业论文 |

## 详细设计

### 操作流程

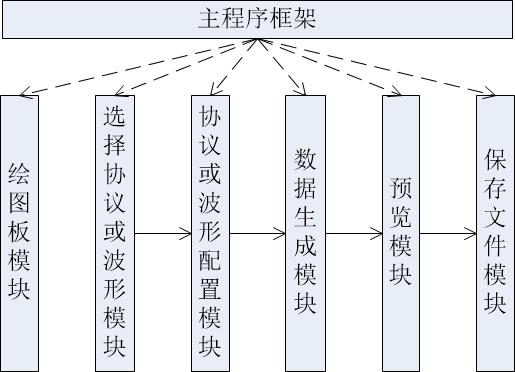
图片 3‑2 为项目的普遍执行流程。



图片 3‑2 操作流程图

### 模块划分

通过分析，获取到五个模块：选择协议或波形模块、协议或波形配置模块、数据生成模块、预览模块、保存文件模块；再加上一个可以支持用户自己绘制波形并按照规定格式保存数据的绘图板模块。图片 3‑3 为系统模块结构图。



图片 3‑3 系统模块图

注：项目是由我和搭档两个人共同开发完成的，由于他是硬件专业而我是软件专业，因此由他负责协议或波形配置模块、数据生成模块、保存文件模块；我负责选择协议或波形模块、预览模块、绘图板模块。

### 选择协议或波形模块设计

选择协议或波形模块的功能是查看指定（plugin）文件夹下的动态链接库文件，将动态链接库的自省信息列出来，供用户选择。基本流程如图片 3‑4。

图片 3‑4 选择协议或波形模块

### 预览模块设计

预览模块是界面部分的重点部分。因为模拟信号与数字信号的巨大差异，它们需要分开处理。

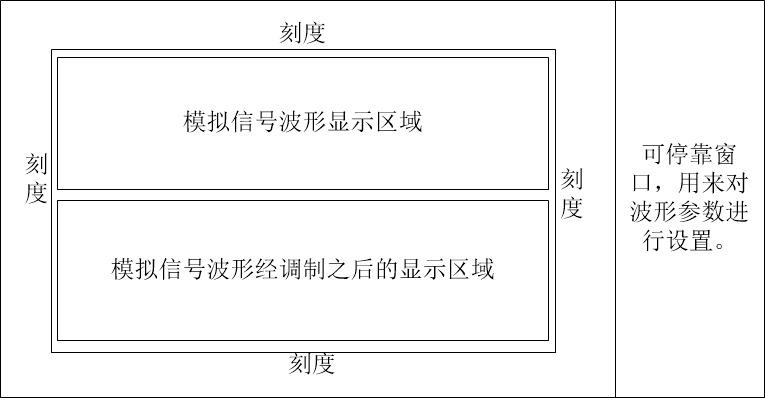
1. 模拟信号预览

模拟信号的预览，需要描绘出波形的起伏，也就是在一个周期内波形的形态。

文件格式要求，使用2048个整数来表示一个波形周期，而且每个整数要求必须在0-1023之间；如何将这2048个点绘制到屏幕上一个指定的矩形区域而不会使波形失真便是模拟波形预览要解决的问题。

项目采用的是线性插值算法：将2048个点插值到将要显示区域中，这样就可以满足显示波形与信号数据一致性的要求。

实际工作中，用户经常希望同时看到模拟信号的波形以及模拟信号经过调制之后的波形，那么为了满足用户的需求，项目中将模拟信号显示的区域分为上下两部分：上面一部分用来显示模拟信号的波形，下面一部分用来显示模拟信号经过调制之后的波形。同时在右侧显示波形的设置项，左侧的界面可以随着右侧的设置项的变化而变化。图片 3‑5是模拟信号预览窗体的草图。

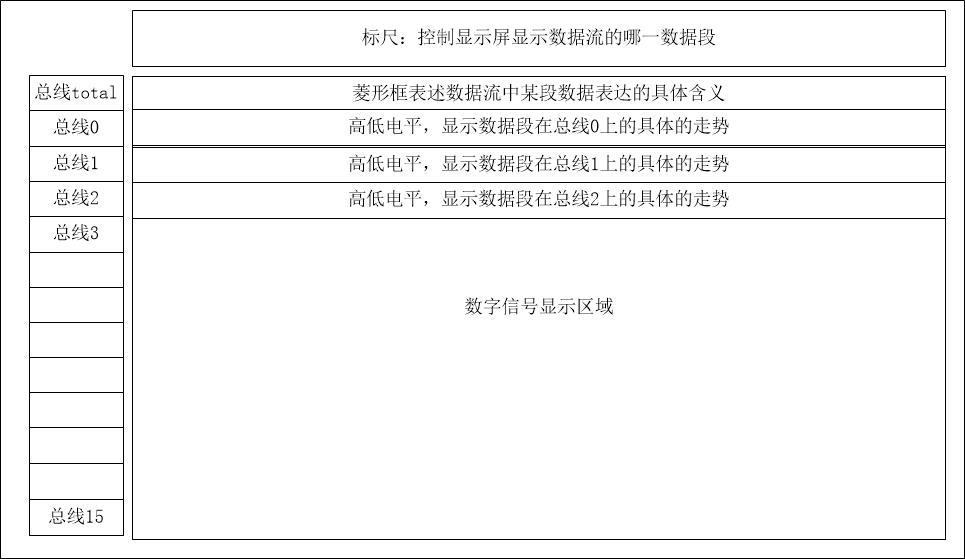


图片 3‑5 模拟信号显示窗体草图

另外，为了实现易用性，满足不同用户的需求，用户可以通过双击左键或选择上下文菜单两种不同的方式来显示设置对话框。

1. 数字信号预览

数字信号是经过处理后的模拟信号，它只有两种表现形式：0（低电平）、1（高电平）。仅仅绘制数字信号的走势是毫无意义的，因为用户并不能通过数字信号的走势来确认这段数据表达的含义，因此项目中使用菱形框来描述在某个协议下一段数据流表达的含义。

与模拟信号不同，数字信号数据流的长度不固定，有些协议下数据流的长度会很长，因此就需要一个滚动条机制来帮助用户确认显示哪一段数据流。图片 3‑6是数字信号显示窗体的草图。

图片 3‑6 数字信号显示窗体草图

相比于模拟信号侧重于表现一个周期内波形的起伏，数字信号则侧重于表现在特定的协议下数据流需要遵循的规范。相同的一串数据流经过不同的数字协议解析后将会表达不同的含义，而数字信号预览模块不可能提前知道每个协议的规范，因此就需要出现一种与协议无关的数据表现方式。

因为用户最关心的是某一段数据流表现的含义，那么可以约定从协议处理模块读取数据时，由协议处理模块额外提供一些信息。以下是从协议处理模块获取的信息：

* 数据块ID：表明这个有含义的数据块在数据流中的位置
* 数据块长度（时间长度）
* 数据块表示的含义（数据、空闲、开始、结束等）
* 用户设置的数据块显示的颜色：仅仅用来显示，帮助用户区分不同数据块

根据实际考察，用户经常需要进行以下操作：跳到数据流开始处、跳到数据流结束处、缩小显示、放大显示。为了满足不同知识背景用户的操作习惯，软件将实现两种操作方式：第一种，通过点击鼠标右键调出上下文菜单，选择用户想要的操作；第二种，通过鼠标与键盘的组合，使用户快速的进行操作。

### 绘图板模块设计

绘图板模块相对于其他模块较为独立。它自己形成一个可执行文件，可以在主程序中作为一个工具打开，也可以单独打开。

绘图板模块的目的是为了生成用户需要的波形。它的输入是用户通过鼠标的“胡乱”滑动，因此可以通过获取用户鼠标按下左键并滑动时的轨迹，来取得输入；然后将这些屏幕上的点通过线性插值算法映射成2048个整数，就得到符合规定的点集合。

### 内部接口设计

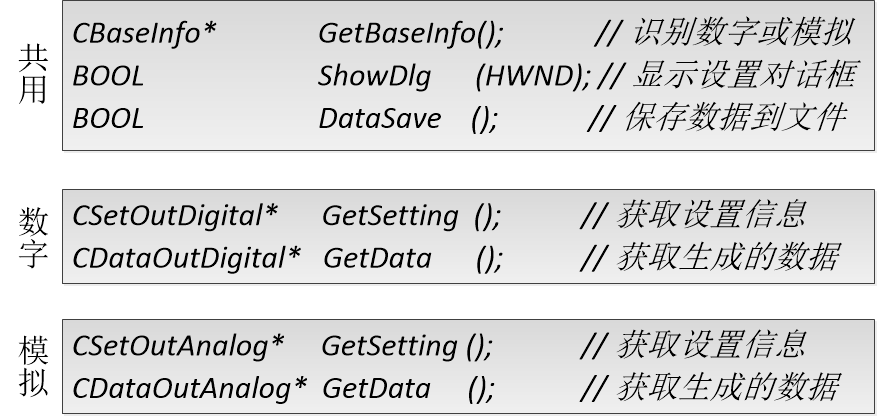
为了实现程序的扩展性，项目中决定将波形或协议设置模块、波形或协议数据生成模块的实现放到动态链接库中，这样其他开发人员只需要按照一定的接口规定，就可以为程序添加波形或协议处理插件。

接口主要定义了主窗体与数据生成模块之间数据的传递：

* 获取基本信息：指明DLL是用来处理数字信号的还是模拟信号的
* 弹出设置窗体：用户可以通过设置窗体来对波形或协议进行参数设置
* 获取设置参数：得到用户设置的显示参数
* 获取生成的数据：得到通过波形或协议参数产生的数据
* 保存数据：将产生的数据保存到文件

# 系统具体实现

## 内部接口的实现

由于数字信号与模拟信号差异较大，因此用于数据信号和用于模拟信号的接口函数不完全一样。图片 4‑1为内部接口函数定义。

图片 4‑1 内部接口实现

其中CBaseInfo、CSetOutDigital、CDataOutDigital、CSetOutAnalog、CdataOutAnalog五个类都是数据类，专门用来存储数据。

## 选择协议或波形模块的实现

选择协议或波形模块由类ListPluginsDialog进行管理。类的定义如代码段 4‑1。

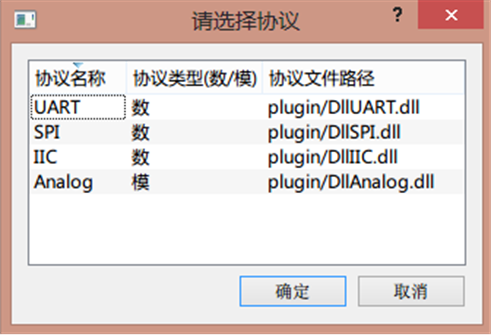
代码段 4‑1 类ListPluginsDialog的定义

|  |
| --- |
| class ListPluginsDialog : public QDialog, private Ui::ListPluginsDialog  {  public:  ListPluginsDialog(QWidget \*parent = 0);  // 获取plugins文件夹下所有符合规定的动态链接库文件  void findPlugins();  private:  // 用来插入数据到对话框以及从对话框获取数据  QStandardItemModel \*model;  signals:  // 该对话框为模态对话框，当选中某个项目并确认时，发射这个信号  void selectedFile( QString,QString,QString );  private slots:  // 当确认按钮被按下时，调用这个函数  void onOkClicked();  }; |

类在构造的时候，设置对话框的显示属性，并连接相应的signal和slot；当该类被告知需要显示选择对话框的时候，它将会在plugin文件夹下查找所有的动态链接库文件（Dll），然后逐一获取该动态链接库文件的自省信息，最后将这些信息插入到对话框中；当用户选择了对话框中的某一项，并点击确认按钮时，该类的onOkClicked函数将会被调用，它负责提取出用户点击项的详细信息，并将这些信息通过selectedFile信号发送出去。代码段 4‑2是关键代码段。

代码段 4‑2 类ListPluginsDialog中findDialog方法的实现

|  |
| --- |
| // 获取plugins目录下的所有dll文件信息  QDir dir(QString("plugin\\"));  QStringList filters;  filters << "\*.dll";  dir.setNameFilters(filters);  QFileInfoList allFiles = dir.entryInfoList ();  // 项目中专门使用下面的类来管理动态链接库文件  PluginInfo \* plugin;  // 遍历获取到的dll文件，它们的自省信息显示出来  for(int i = 0;i<allFiles.size ();i++)  {  QFileInfo fileInfo = allFiles.at (i);  // 初始化PluginInfo  plugin = new PluginInfo(UKNOWN\_TYPE);  if(plugin->setName(fileInfo.filePath()))  {  if(plugin->loadBaseInfo())  {  // 调用GetBaseInfo函数来获取自省信息  CBaseInfo \* bi = plugin->getBaseInfo();  // 在对话框中增加一行空白行  model->insertRow(0);  // 在新增的空白行中插入自省信息  model->setData(  model->index(0,0),  QString(gbk\_codec->toUnicode(bi->strName))  );  model->setData(  model->index(0, 1),  (bi->isDigital == false)?tr("模"):tr("数")  );  model->setData(model->index(0, 2), fileInfo.filePath ());  }  }  delete plugin;  } |

该模块最终实现的界面如图片 4‑2。

图片 4‑2 选择协议或波形模块的窗体实现

## 模拟信号预览模块的实现

模拟信号的显示使用了多个类的协同工作来实现。首先，使用类AxisOfCoordinates负责绘制模拟波形以及模拟波形经过调制之后的波形，代码段 4‑3是类AxisOfCoordinates的定义。

代码段 4‑3 类AxisOfCoordinates的定义

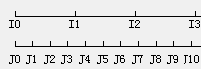
|  |
| --- |
| class AxisOfCoordinates : public QWidget  {  public:  AxisOfCoordinates( QWidget \*parent = 0);  PluginInfo \* pi;  protected:  // 重新实现上下文菜单事件  void contextMenuEvent (QContextMenuEvent \*);  // 重新实现鼠标双击事件  void mouseDoubleClickEvent(QMouseEvent \*);  // 重新实现绘图事件  void paintEvent (QPaintEvent \*);  private:  // 上下文菜单  QMenu \* menu;  QAction \* disSettingDialog;  // 坐标轴上方距离屏幕上方的距离  const int upDownSpace;  // 坐标轴左边距离屏幕左边的距离  const int leftRightSpace;  // 坐标轴距离绘制波形区域的距离  const int circle;  // 用来保存坐标轴刻度值的数组  QVector<QString> xCoor;  QVector<QString> yCoor;  // 用来保存波形数据的数组  typedef QVector<int> WaveVector;  WaveVector waveVector;  // 从波形处理模块获取的波形数据  CDataOutAnalog \* data;  // 从波形处理模块获取的波形设置参数  CSetOutAnalog \* set;  // 绘制网格：坐标轴线、背景网格、刻度值  void paintMesh(QPainter \* pt);  // 根据waveVector的值绘制一个波形  void drawWave( QPainter \*pt,QRect,QColor );  public slots:  // 获取模拟信号的数据和设置  void readData();  }; |

在类AxisOfCoordinates的构造函数中，进行了必要的变量初始化以及signal与slot的连接；初始情况下，它没有与任何波形类型相关联，因此在绘图函数paintEvent中只是绘制了必要的坐标轴以及背景；当它与具体的波形相关联时，它会被告知调用readData函数，readData函数将会从管理动态链接库的类PluginInfo中获取模拟信号的数据和设置参数，进行必要的处理，然后通知绘图函数paintEvent绘制坐标轴以及波形。

其中，从动态链接库中获取的数据，是2048个int值，每个int值都介于0~1023之间，如何将这2048个数据映射到屏幕上而不使波形失真是技术难点。

项目中采用的是线性插值算法。假设屏幕上显示波形的区域宽度为x个像素点（x的值一般为100~1000），那么这2048个点落到屏幕上显示波形的区域后，很多点都落到了屏幕上相邻两个像素点的中间（即落点像素点不为整数）。如图片 4‑3，假设J0-J10是2048个点中的11个点，I0-I3是x个点中的4个点；其中J0经过映射之后落到了J0上，J1、J2、J3经过映射之后落到了J0与J1之间，J4、J5、J6落到了I1、I2之间…；现在的问题变成了如何通过已知的Jn（n=0…10）的值求取Im（m=0…3）的值。要求取Im的值，首先需要找出一个k，使得Jk与Jk+1落到Im的两侧；设Im对应的像素点位置为X0（整数），对应的值为Y0，Jk对应的像素点位置为X1（非整数），对应的值为Y1，Jk+1对应的像素点位置为X2（非整数），对应的值为Y2,则Im对应的值为：

Y0 = Y2 - （Y2-Y1）\*（X2-X0）/(X2-X1)。



图片 4‑3 逻辑点与像素点的对应图

代码段 4‑4是实现该算法的代码段。

代码段 4‑4 类AxisOfCoordinates中线性插值算法的实现

|  |
| --- |
| void AxisOfCoordinates::drawWave ( QPainter \*pt,QRect rect,QColor color)  {  // 局部变量初始化  double x = 0.0;  double y = 0.0;  int oldx = -10000;  int oldy = -10000;  // 绘图路径  QPainterPath path;  // 保存之前画笔，用来在程序结束的时候进行恢复  pt->save();  QPen pen(QBrush(QColor(color)),2);  pt->setPen(pen);  // 使用线性插值算法将2048个点映射到屏幕上  for( int i = 0 ; i < waveVector.count() - 1 ; i ++ )  {  // 找到2048个点中临近屏幕上点的两个连续点  if( static\_cast<int>(  static\_cast<double>(i) \* rect.width() / waveVector.count()  ) < static\_cast<int>(  static\_cast<double>(i+1)\* rect.width() / waveVector.count() )  )  {  // 求出屏幕上该点的插值后的值  x = static\_cast<double>(i+1) \* rect.width() / waveVector.count();  x = static\_cast<double>( static\_cast<int>(x) );  y = static\_cast<double>(waveVector.at(i)) +  ( x - static\_cast<double>(i) \* rect.width() / waveVector.count() )\*  ( waveVector.at(i+1) - waveVector.at(i) ) /  static\_cast<double>( rect.width() ) / waveVector.count();  // 在绘图路径上绘制点  if(!( oldx == -10000 && oldy == -10000 ))  {  path.moveTo(oldx,oldy);  path.lineTo(  static\_cast<int>(x)+rect.x(),  rect.y()+rect.height()  - static\_cast<int>(y\*rect.height()/1023));  }  oldx=static\_cast<int>(x)+rect.x();  oldy=rect.y()+rect.height()-static\_cast<int>(y\*rect.height()/1023);  }  }  // 将点绘制到屏幕上  pt->drawPath(path);  pt->restore();  } |

之后，通过类AxisOfCoordinates和DockWidget组合成类SimulatePreview，类SimulatePreview直接与程序主体框架类MainWindow进行交互，然后控制类AxisOfCoordinates对用户的操作进行响应；类AxisOfCoordinates不会与类MainWindow直接进行交互，实现了类的封装。

注：因为技术原因，项目中没有实现停靠窗口这一设计，而由于时间的问题，我们只能将设置对话框显示成一个模态对话框，只在修改参数的时候进行显示。

该模块最终实现的界面如图片 4‑4。

图片 4‑4 模拟信号显示模块的窗体实现

注：由于设置对话框并非我实现的，因此这里不做过多描述

## 数字信号预览模块的实现

数字信号的显示也使用了多个类的协同工作来实现。首先，使用类Staff来控制显示波形的哪一段（数字信号数据流将要被显示的起点以及宽度），类ButtonTree用来绘制总线，类DigitPreviewArea类负责根据Staff提供的位置信息来绘制波形。代码段 4‑5是这三个类的定义。

代码段 4‑5 类Staff、ButtonTree、DigitPreviewArea的定义

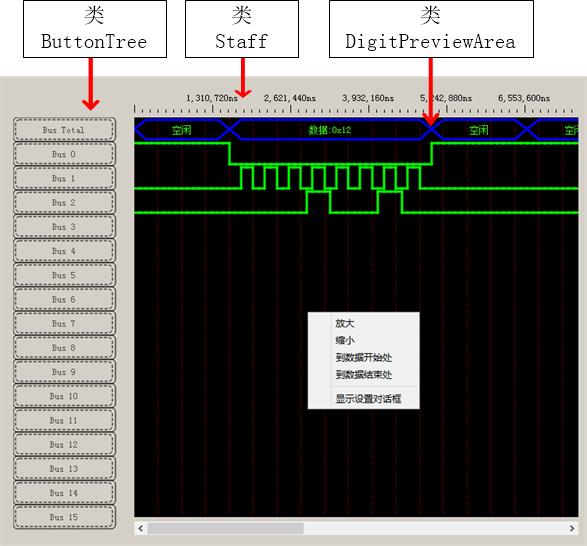
|  |
| --- |
| class Staff : public QWidget  {  public:  Staff(QWidget \*parent = 0,int startTime = 0,int time = 128);  protected:  void paintEvent (QPaintEvent \*);  void wheelEvent (QWheelEvent \*);  private:  quint64 startTime; //标尺标注的开始时间，单位为1ns  int time; //标尺中每一小格表示的时间，单位为1ns  int maxDisNum; //标尺显示的小格个数  int SmallLength; //每小段标尺的像素距离  int NumerSmallLength; //每大段标尺的小段个数  signals:  // 当标尺的宽度、起始点、标尺小格所表示的时间改变时，信号被发送  // startTime 表示标尺最左端表示的时间（ns）  // timeLength 表示标尺上一小格表示的时间（ns）  // disLength 表示标尺上一小格表示的像素长度（px）  // disNum 表示标尺上显示的小格数量  void disChanged(  quint64 startTime,  int timeLength,  int disLength,  int disNum  );  public slots:  //将其他widget的鼠标滚轮事件转发到该类鼠标滚轮事件处理函数中  void wheelEventOnOtherWidget( QWheelEvent \* );  void timeBeHalve(); // 设置time变成原来的一半  void timeBeDouble(); // 设置time变成原来的两倍  void startTimeAdd(int); // 使得startTime增加或减少  };  class ButtonTree : public QWidget  {  public:  ButtonTree(QWidget \*parent = 0);  private:  // 标识bus按钮所在的位置  QVector<QRect> buttonPos;  // 判断当前的点击位置是不是在一个bus按钮上，若是，返回bus按钮ID  int isEventPosInOneButton(int,int);  // 当前选中的bus按钮  int currentSelectedButton;  protected:  // 重写绘图函数  void paintEvent (QPaintEvent \*);  // 重写鼠标按下消息  void mousePressEvent (QMouseEvent \*);  signals:  // 当某个bus按钮被选中后时，发射这个信号  void buttonSelected(QRect);  // 当按钮位置变化时，发送这个信号  void buttonPosChanged(QVector<QRect>);  };  class DigitPreviewArea : public QWidget  {  public:  DigitPreviewArea(QWidget \*parent = 0);  PluginInfo \*pi;  // 上下文菜单  QMenu \* menu;  QAction \* disSettingDialog;  QAction \* bigAction;  QAction \* smallAction;  QAction \* toStartAction;  QAction \* toEndAction;  protected:  // 重写上下文菜单事件  void contextMenuEvent (QContextMenuEvent \*);  // 重写绘图事件  void paintEvent(QPaintEvent \*);  // 重写鼠标滚轮事件  void wheelEvent (QWheelEvent \*);  // 重写鼠标双击事件  void mouseDoubleClickEvent(QMouseEvent \* event);  private:  // 标识当前被选中的bus位置  QRect currentSelectedButtonPos;  // 标识所有bus的位置  QVector<QRect> buttonPos;  // 通过以下变量绘制波形  quint64 startTime;  int timeLength;  int disLength;  int disNum;  // 保存数字信号的数据和设置  CDataOutDigital \* data;  CSetOutDigital \* set;  // 负责构造上下文菜单  void createContextMenu();  //负责绘制bus的0号总线，也就是16根总线数据的混合体，菱形框  void drawBusTotal( QPainter \* );  // 负责绘制每一个菱形  void drawRhomb( QPainter \*,quint64,int,QString,QColor );  //负责绘制bus的1-16号总线，折线  void drawBus( QPainter \* );  //负责绘制折线的参考线，每两根参考线之间的距离代表一个二进制位的时间  void drawReferrenceLine( QPainter \* );  signals:  //当鼠标滚轮在preview上滚动时（同时按下ctrl键），发射这个消息  void wheelEventHappened(QWheelEvent \*);  // 当鼠标滚轮在preview上滚动时（没有按下ctrl键），发射这个消息  void wheelRoll(int);  // 当鼠标双击事件发生时，发射这个消息  void mouseDoubleClickEventHappened();  // 当读取数据完毕之后，发射这个消息  void dataInfo(int,int);  public slots:  // 即时获取当前选中的bus位置  void buttonSelected(QRect);  // 即时获取bus位置的变化  void buttonPosChanged( QVector<QRect> );  // 即时获取当前需要显示时间段  void disChanged(quint64,int,int,int);  // 获取数字信号的数据和设置  void readData();  }; |

类Staff中变量：startTime、time、maxDisNum、SmallLength来控制数字信号显示区域的一些属性，startTime控制显示区域最左侧显示的数据的时间（单位为ns），time表示标尺中一个最小的刻度显示的时间长度（单位为ns），maxDisNum表示显示区域中最多能显示的刻度数，SmallLength表示每个刻度在屏幕上显示的像素长度；通过这四个变量，可以控制数字信号数据流中哪一段在屏幕上显示、以及显示到屏幕上的那个位置（x轴的位置）。

类ButtonTree用来绘制总线按钮，它用来界定每条总线的高度。

类DigitPreviewArea用来绘制数字信号将要被显示的数据段的走势。与类AxisOfCoordinates的实现类似，类DigitPreviewArea在构造函数中对一些变量进行初始化，以及建立signal与slot的连接；在未与具体协议绑定之前，绘图函数只是绘制出背景；当类DigitPreviewArea被告知与某一个具体协议绑定之后，类DigitPreviewArea将会通过PluginInfo从动态链接库中获取出数字信号数据流以及数据信号显示参数，然后绘图函数根据这些数据，调用 drawBusTotal（负责绘制bus的0号总线，也就是16根总线数据的混合体，菱形框）、drawBus(负责绘制bus的1-16号总线，折线)、drawReferrenceLine(负责绘制折线的参考线，每两根参考线之间的距离代表一个二进制位的时间)来绘制用户想要观察的数据流走势图。

由于直接控制这三个类进行数字信号的显示较为复杂，而且为了在提供服务时屏蔽掉细节问题，项目中使用类DigitPreview来对这三个类进行管理：DigitPreview直接与MainWindow进行交互，然后通过控制Staff、ButtonTree、DigitPreviewArea来实现用户的操作请求，而Staff、ButtonTree、DigitPreviewArea对MainWindow来说是透明的。

该模块最终实现的界面如图片 4‑5。

图片 4‑5 数字信号预览模块的窗体实现

注：由于设置对话框并非我实现的，因此这里不做过多描述

## 绘图板模块的实现

绘图板模块较为独立，它的最终表现形式是一个单独的可执行文件，也就是说它除了在主程序中打开外，还可以单独打开。

绘图板模块的主要功能由类Canvas实现。代码段 4‑6是Canvas的定义。

代码段 4‑6 类Canvas的定义

|  |
| --- |
| class Canvas : public QWidget  {  public:  Canvas(QWidget \*parent = 0);  // 文件中的数据  QVector<int > data;  // 绘图板上的点，由data插值获取  QVector<QPoint> path;  // 文件中数据与屏幕上的点相互转换  void pathToData();  void dataToPath();  // 插值算法  void LinearInterpolation( QRect,QVector<QPoint>,QRect,QVector<QPoint> & );  protected:  void paintEvent (QPaintEvent \*);  void mousePressEvent (QMouseEvent \*);  void mouseReleaseEvent (QMouseEvent \*);  void mouseMoveEvent (QMouseEvent \*);  private:  // 绘图板绘图区的边界  const int leftright;  const int updown;  // 识别当前鼠标左键是否被按下  bool isPushDown;  }; |

Canvas在构造函数中，初始化一些变量以及signal与slot的连接；当用户控制鼠标左键按下时，isPushDown被置为true；当用户鼠标移动时，若是isPushDown为true，则记录鼠标移动的轨迹，并将轨迹中在界定区域内的点记录到path变量中；当用户鼠标左键抬起时，Canvas将会把path中的数据通过插值算法映射到data中。

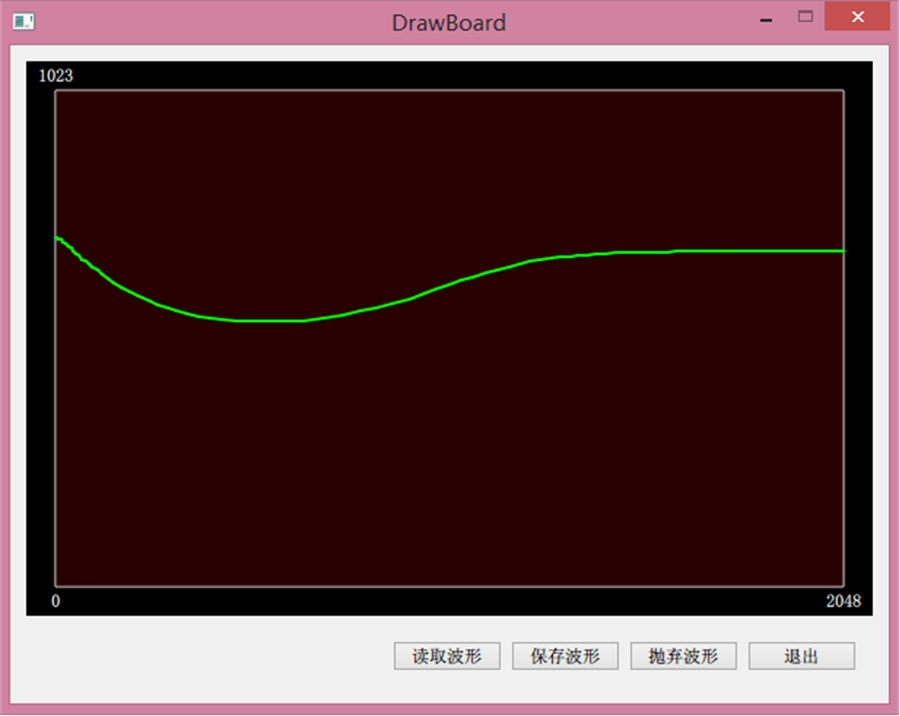
为了将数据处理与界面显示想分离，软件使用了另外一个类DrawBoard来控制Canvas。代码段 4‑7是DrawBoard的定义。

代码段 4‑7 类DrawBoard的定义

|  |
| --- |
| class DrawBoard : public QWidget, private Ui::DrawBoard  {  public:  DrawBoard(QWidget \*parent = 0);  public slots:  // 响应读文件的操作  void readWave();  // 响应保存数据到文件的操作  void saveWave();  // 响应清除当前数据的操作  void clearWaveData();  }; |

类DrawBoard用来与用户交互，控制Canvas完成用户的操作请求。

该模块最终实现的界面如图片 4‑6。



图片 4‑6 绘图板模块的窗体实现

## 程序主体框架的实现

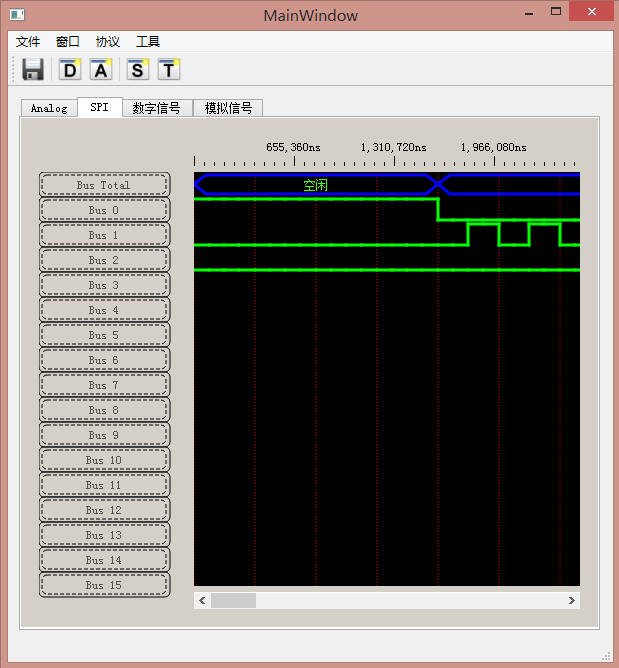
程序的主体框架，由类MainWindow控制，负责控制以上的模块来帮助用户解决实际问题。主程序使用属性页的方式加载数字信号处理模块以及模拟信号处理模块；当主程序框架接收到来自用户的操作请求后，将会根据当前页面的信息来决定将这些操作请求传递至数字信号处理模块还是模拟信号处理模块；信号处理模块获取到这些操作请求之后，将会对这些请求进行响应，以实现用户的需求。

代码段 4‑8是MainWindow的类定义。

代码段 4‑8 类MainWindow的定义

|  |
| --- |
| class MainWindow : public QMainWindow, private Ui::MainWindow  {  public:  MainWindow(QWidget \*parent = 0);  protected:  void closeEvent (QCloseEvent \*);  private:  // 控制绘图板工具的显示  ExtensionTool \* et;  signals:  // 保存按钮被按下时，发射这个信号  void saveData();  // 通知被选中的tab页进行初始化工作  void initWidget(QString);  private slots:  // 将动态链接库文件跟tab页绑定  void protocalLinktoTab(QString,QString,QString);  // 增加一个新的处理数字信号的tab页  void addDigitWindow();  // 增加一个新的处理模拟信号的tab页  void addSimulateWindow();  // 当点击选择协议对话框时，调用这个函数  void selectProtocal();  // 移除一个tab页，并release掉与这个tab页绑定的动态链接库  void removeOneTab(int);  // 发射给当前tab一个保存数据到文件的信号  void onActionSaveData();  // 打开绘图板工具  void openDrawBoard();  }; |

该模块最终实现的界面如图片 4‑7。



图片 4‑7 主体框架的窗体实现

其中菜单包括：保存数据到文件、打开数字信号处理窗口、打开模拟信号处理窗口、打开选择协议或波形对话框、打开绘图板工具。MainWindow中的slot就是为了响应这些菜单操作：onActionSaveData用来响应保存数据到文件操作，addDigitWindow用来响应打开数字信号处理窗口操作、addSimulateWindow用来响应打开模拟信号处理窗口操作、selectProtocal用来响应打开选择协议或波形对话框操作、openDrawBoard用来响应打开绘图板工具操作。

# 总结

项目中通过使用MVC架构，模块划分明确，符合高内聚、低耦合的特性，使得分工更为明确，程序逻辑更为清晰，程序的模块重用性大大提高，程序也变得更加具有扩展性。而在开发过程中，严格的遵循软件开发流程，按照需求获取、设计、编码的思路，规避了大部分因管理不善而导致项目失败的可能性。总体来说，项目实现了最初的需求，开发出了一款可以帮助用户生成自己需要的波形数据的软件。

但是，无论是在项目开发过程中还是在最终的项目实现上，都出现了一些问题：

* 因为管理经验欠缺而导致的时间进度安排不合理，过低的估计了学习新知识的时间代价，导致项目逾期完成，且一些问题无法解决。
* 过低的估计了UI界面设计的难度，只是站在开发人员的角度设计操作习惯而没有考虑其他不同类型用户的使用习惯。
* 波形设置对话框无法停靠到主窗体上，使得用户随时查看当前波形设置；而只能作为一个模态对话框，当需要设置的时候显示，需要绘制波形的时候隐藏。
* 即使使用了线性插值算法算法，但在实际操作过程中，绘图板模块保存到文件的数据被重新读取后，会出现锯齿。

本次项目的开发人员经验较为欠缺，是一次初步尝试，在以后将会有经验丰富的开发人员指导并重启项目，以真正的达到商用软件的标准。