课程编号：0008190

计算机硬件类综合性课程设计
设计报告

(2021-2022-2)

题　　目：　逻辑分析仪

学部（院）：　樊恭烋荣誉学院

专业班级：　樊恭烋荣誉学院 190004

报 告 人：　19071001　唐仡夫

考勤(10 分)	设计实现(70 分)	文档(20 分)	总评(100 分)
评语：			
教师签字		评阅日期	2022.10.

一、概述

1.1 选题意义

本课设以总线时序为切入点，选择 I2C 协议进行逻辑分析。考虑到现行的逻辑分析仪大多使用采样与分析分开的方式进行，这种设计方式可以提供极大的处理速度、适配性以及功能性，但是这种方式并不便于携带或轻量型的调试工作。我们基于 Arduino 设计了一体式逻辑分析仪，提供更加便捷的显示、分析方式。

1.2 作品功能概述

本作品可以实现 I2C 总线采样、数据显示以及结果分析，并提供其它辅助功能：

1. 对 I2C 信号进行采样（1khz 采样 4096 个数据点）
2. 可以在屏幕中显示原始信号
   a) 触摸屏操作左右滑动
   b) 缩放屏幕
3. 对 I2C 信号进行解析，并显示基本信息
   a) 显示开始地址
   b) 显示地址
   c) 显示 R/W
   d) 显示数据长度
   e) 显示数据以及 ACK/NACK，可以使用触摸屏操作上下滑动
4. 搜索 I2C 信号
5. 其它功能
   a) 屏幕刷新
   b) 调整触摸屏

二、总体设计

本项目由 Arduino 开发板，显示模块以及待采总线组成。Arduino 开发板负责从待采总线中采集待测信号，并进行分析。显示模块负责显示采集、分析结果，并允许用户通过触摸屏与系统进行交互。显示模块有两种模式，分别为微雪官方的 TFT 显示器模块以及由我们设计的显示驱动模块于裸屏组成的显示模块。

外部设备由电脑以及 I2C 转 USB 调试仪构成，调试仪与电脑通过调试软件连接，调试仪与 Arduino 开发板通过 I2C 总线连接，该总线作为待测总线进行信号采集。

我们通过电脑中的调试软件控制调试仪，向 Arduino 发送可控的数据，通过 Arduino 开发板进行采集、分析，并将分析结果与发送的数据进行比对，以此验证系统的有效性与正确性。

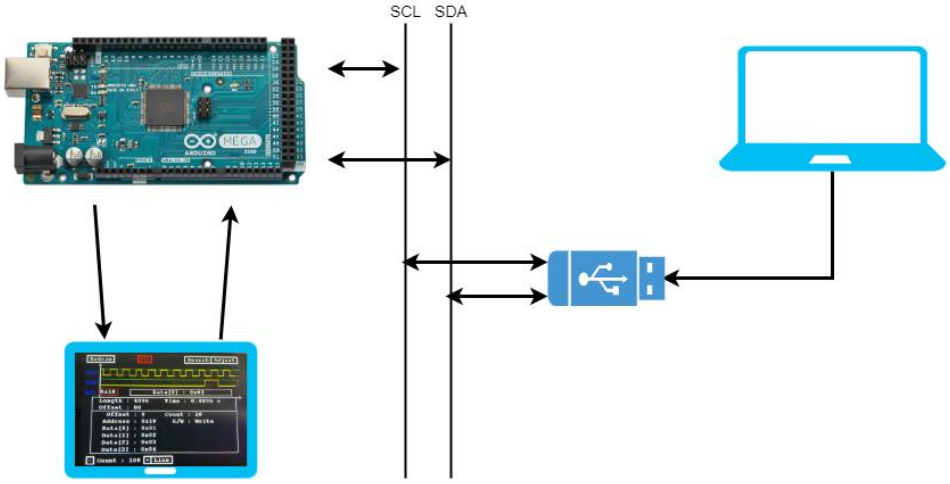

图 1 系统框架图

三、硬件详细设计

3.1 硬件系统结构

本系统主要由三部分组成，即 Arduino 开发板、显示模块以及待采总线，其中显示模块由驱动 PCB 以及显示屏组成，显示屏使用内置 ILI9486 芯片驱动，并通过 37 针脚的 FPC 与驱动 PCB 相连，驱动 PCB 负责讲 SPI 串行数据转换为 16 位并行数据，从而控制显示屏，同时通过读取显示屏电压信息，使用 XPT2046 获得屏幕触控信息，并接入 SPI 总线。显示模块与 Arduino 开发板通过 SPI 总线连接，同时 Arduino 开发板通过多个引脚向显示模块发送控制信号，例如 CS、RST 等信号。待采总线由 I2C 转 USB 调试仪以及电脑组成，调试仪与电脑通过 USB 连接，电脑上的调试程序允许对调试仪进行相关配置，并在调试仪中搭建 I2C Master，使得调试仪通过 I2C 总线与 Arduino 开发板进行通信，从而构成了待采总线。

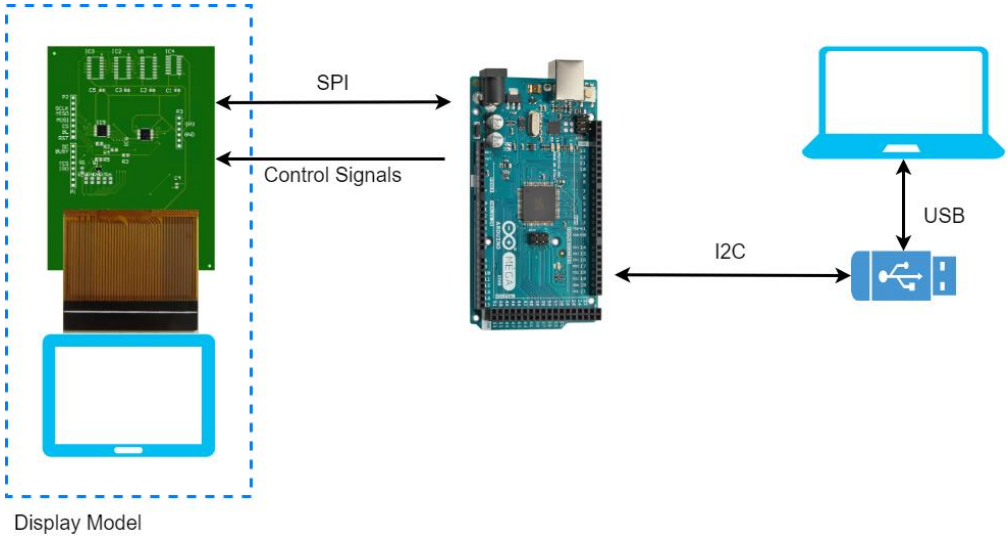

图 2 硬件系统结构图

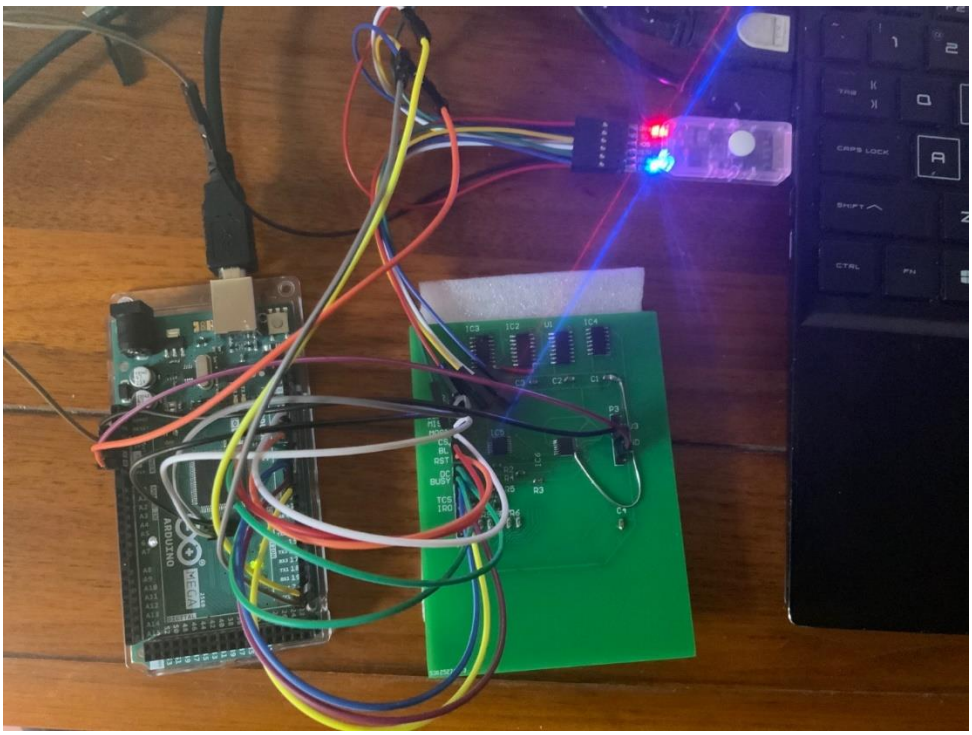

图 3 硬件连接实物图

3.2 核心板

3.2.1 选型

考虑到程序需要大量的 IO 端口以及需要使用大量内存进行采样数据的保存，因此我们对开发板对的 IO 数量、IO 速率以及内存大小有着更高的要求，因此本程序使用 Mega 2560 开发板进行开发。同时 Mega 2560 拥有远超 Uno 开发板的计时器数量，Uno 开发板只有 Timer1，Timer2 以及 Timer3 三个定时器，而 Mega2560 拥有 Timer1， Timer2， Timer3， Timer4，Timer5，Timer6 六个定时器，可以在不影响系统时钟以及 PWM 设置的情况下进行其它频率中断的设置。

Mega 2560 是 Arduino 官方开发的开发板，使用 ATmega2560 与 ATmega16U2 微处理器。Mega2560 拥有 54 个数字管脚、16 个模拟管脚以及 4 个 UART 接口，并提供 USB、ICSP 接口。

ATmega2560 微处理器，其时钟频率为 16MHZ，倍频至 16MIPS，拥有 4KBs EEPROM，8KBs SRAM，32 个 8 位通用寄存器。ATmega16U2 微处理器，时钟频率也为 16MHZ，倍频至 16MIPS，拥有 16KBs ISP 闪存，0.5KBs EEPROM 与 0.5KBs SRAM。

经过测试 IO 速率为 512 分频，即 31.25khz，这一点与其它 Arduino 开发板相同。

Mega2560 的内存大小为 8kb，允许程序开辟 int 数组最长 4096 位，这是由于在 Arduino 编译程序中加入了数组大小限制的结果。

Mega2560 使用外置 5V 电源，允许通过 VIN、USB、PWRIN 方式输入，再经过稳压后，输入到微处理器以及其它组件中，并通过 LP2985 变压芯片，输出 3.3V 电压。

3.2.2 模块原理图介绍

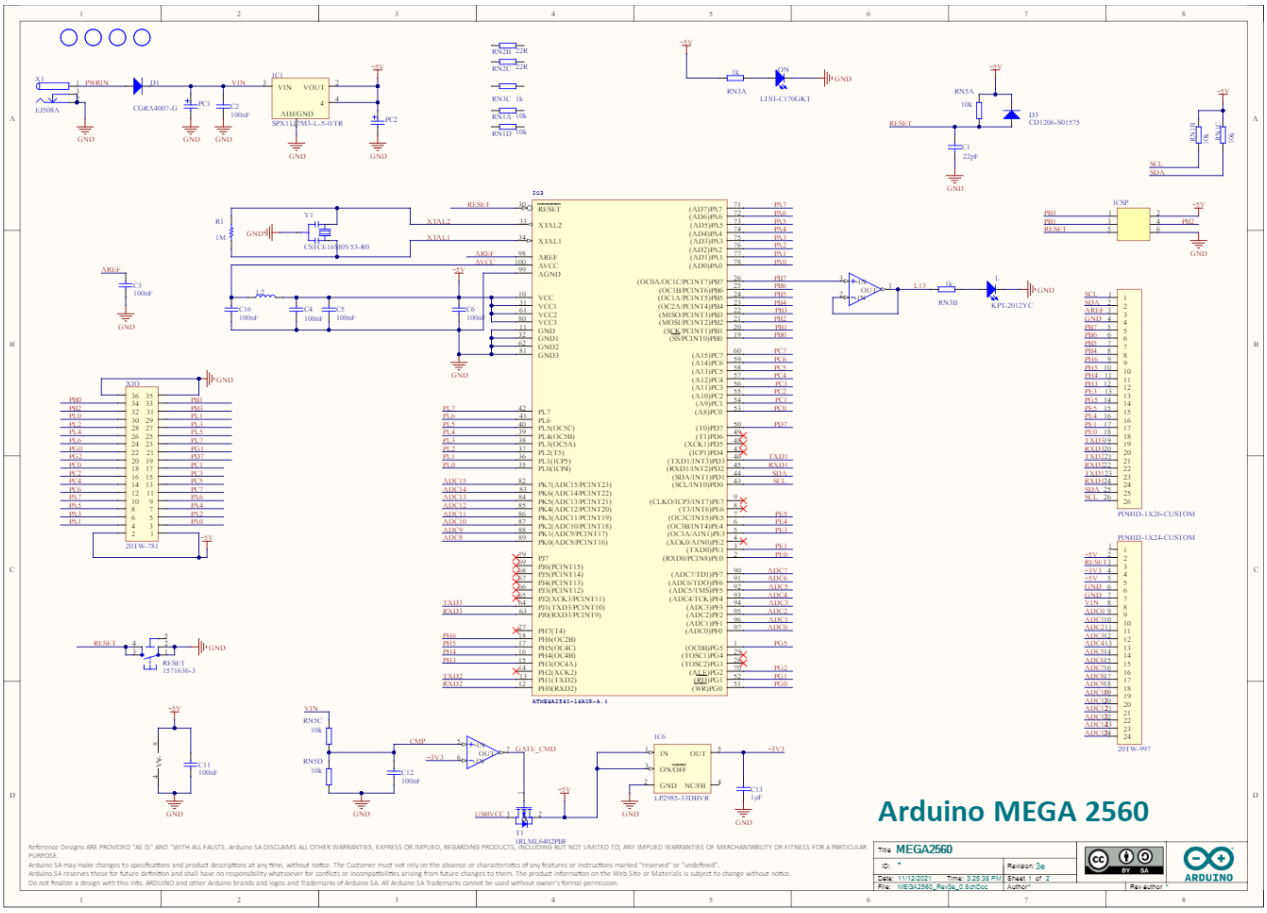

图 4 Mega2560 原理图

图中 IC6 为 5V 转 3.3V 变压芯片，IC3 位 ATMAGA2560 微控制器。

3.2.3 模块 PCB 介绍

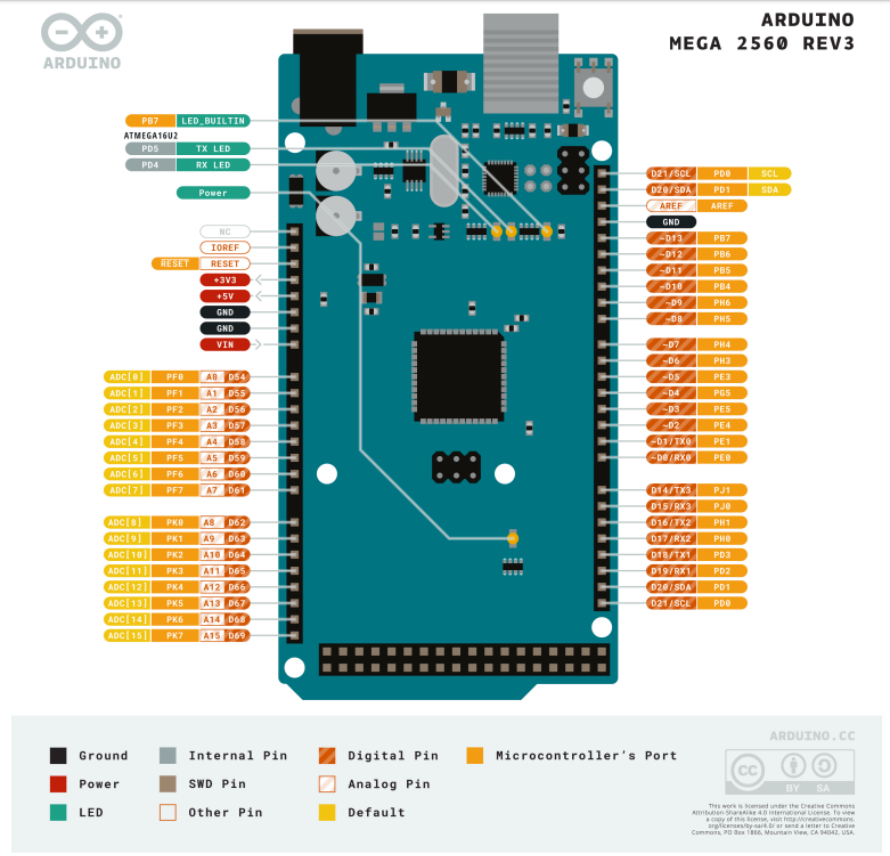

图 5 Mega 2560 引脚图 详情见附录（图 30）

Mega2560 拥有 3.3V 以及 5V 两种电压输出，其数字引脚默认输出为 5V，但是会随着搭载模块的增加，引脚的电压会有所下降。20 与 21 针脚对应 I2C 总线的 SDA 与 SCL。

3.3 I2C 转 USB 调试器

本文中使用 I2C 转 USB 调试器，其具有 USB 插头，可以与电脑进行通信或只进行充电。其包含 SPI 接口与 I2C 接口，可以调试 SPI 总线或 I2C 总线。对于 I2C 总线，其提供 3.3V, 5V, SCL, SDA, FUN 以及 GND 六个引脚。在程序中我们只使用 5V, SCL, SDA 以及 GND 四条引脚即可。

调试器允许用户设置 I2C 主/从模式，并可设置传输速率为 1K,10K,20K,50K,100K 甚至 1M，但是由于 Mega2560 的 IO 速率限制，我们选取了最低传输速率 1K。

用户可在电脑端通过虚拟穿孔连接调试器，使得调试器作为 I2C 向总线定时/非定时发送一条或多条数据。

3.4 显示屏

我们选用 4 寸 Arduino 电阻屏作为显示屏使用，其使用 ILI9486 芯片进行驱动，由于我们使用的是模块屏，其引脚数量以及连接方式与裸屏完全不同，具体的端口说明见显示驱动模块。

3.5 显示驱动模块

我们参考微雪官方的显示驱动模块，针对 Mega2560 进行了些许优化，包括对 5V 转 3.3V 电路的优化以及删除 SD 卡驱动电路等冗余电路。

驱动显示模块通过读取由 Arduino 开发板以 SPI 总线形式发送而来的串行控制信息，并结合 CS，RST 等控制信号，以并行方式输入到显示屏当中。并通过读取显示屏返回的电压信息，计算触摸屏的相关信息，使用 SPI 总线向 Arduino 回传触摸屏信息。

3.5.1 显示驱动模块的原理图设计

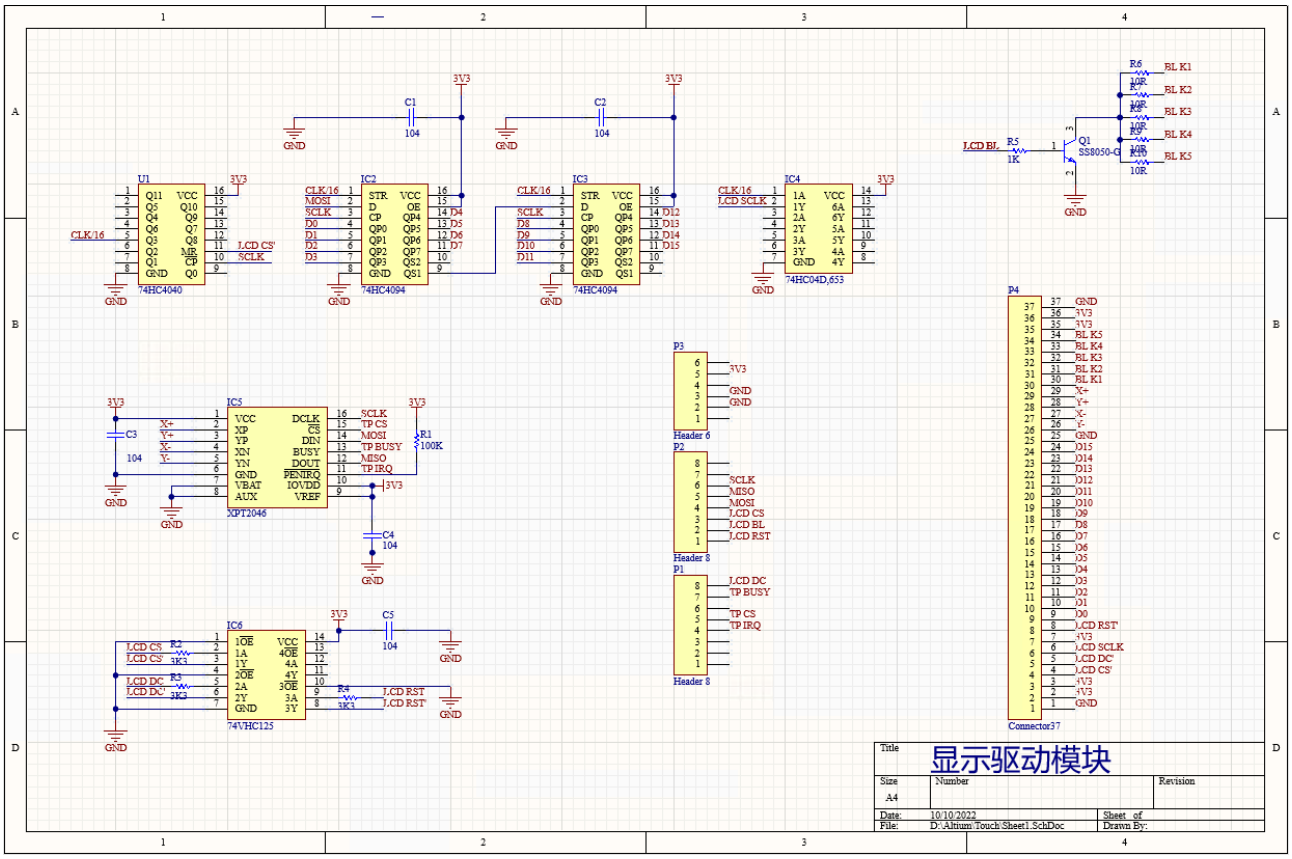

图 6 显示驱动模块原理图

显示驱动模块主要分为五部分，与 Arduino 连接的引脚部分、与显示屏连接的引脚部分、SPI 串行转并行部分、XPT2046 驱动部分以及背光部分。

1. Arduino 引脚部分，由 P1，P2，P3 插排构成，从 Arduino 引入 3.3V 电源，地线，SPI 总线（SCLK,MOSI,MISO 三条引脚）、用于 ILI9486 芯片控制的 LCD_CS、LCD_RST、LCD_DC 引脚，用于 XPT2046 控制的 TP_BUSY、TP_CS、TP_IRQ 引脚，以及用于背光控制的 LCD_BL 引脚。对于控制信号，会经过 74VHC125 芯片缓冲，才会输入到模块当中。
2. 与显示器连接的引脚部分，显示器由 37 针脚的 FPC 与显示驱动模块连接，其中 1、25、37 引脚接地，2、3、5、35、36 引脚接 3.3V 电源，4、5、8 引脚接 ILI9486 控制信号，6 引脚接 16 分频的 SCLK 信号，9 到 24 引脚为 SPI 总线串行转并行后的数据线路，26 到 29 引脚为电容屏电压信号用于 XPT2046 触摸检测，30 到 34 引脚为背光引脚，通过输入不同的电压来改变显示屏的背光亮度。
3. SPI 串行转并行部分，由 74HC4040 分频器以及两块 74HC4094 八位寄存器构成的串行转十六位并行器构成，并且十六分频信号会经过 74HC04D 反相器再输入到显示器中的 ILI9486 芯片中。
4. XPT2046 驱动部分，由 XPT2046 芯片组成，通过读取电阻屏的电压信息，在 Arduino 控制信号的控制下，从 SPI 总线中向 Arduino 传输触摸信息。
5. 背光部分由 NPN 三极管构成，通过背光信号控制 NPN 三极管导通状态，依次控制输入到显示屏中背光引脚的电压，以达到控制背光亮度的作用。

3.5.2 显示驱动模块的 PCB 设计

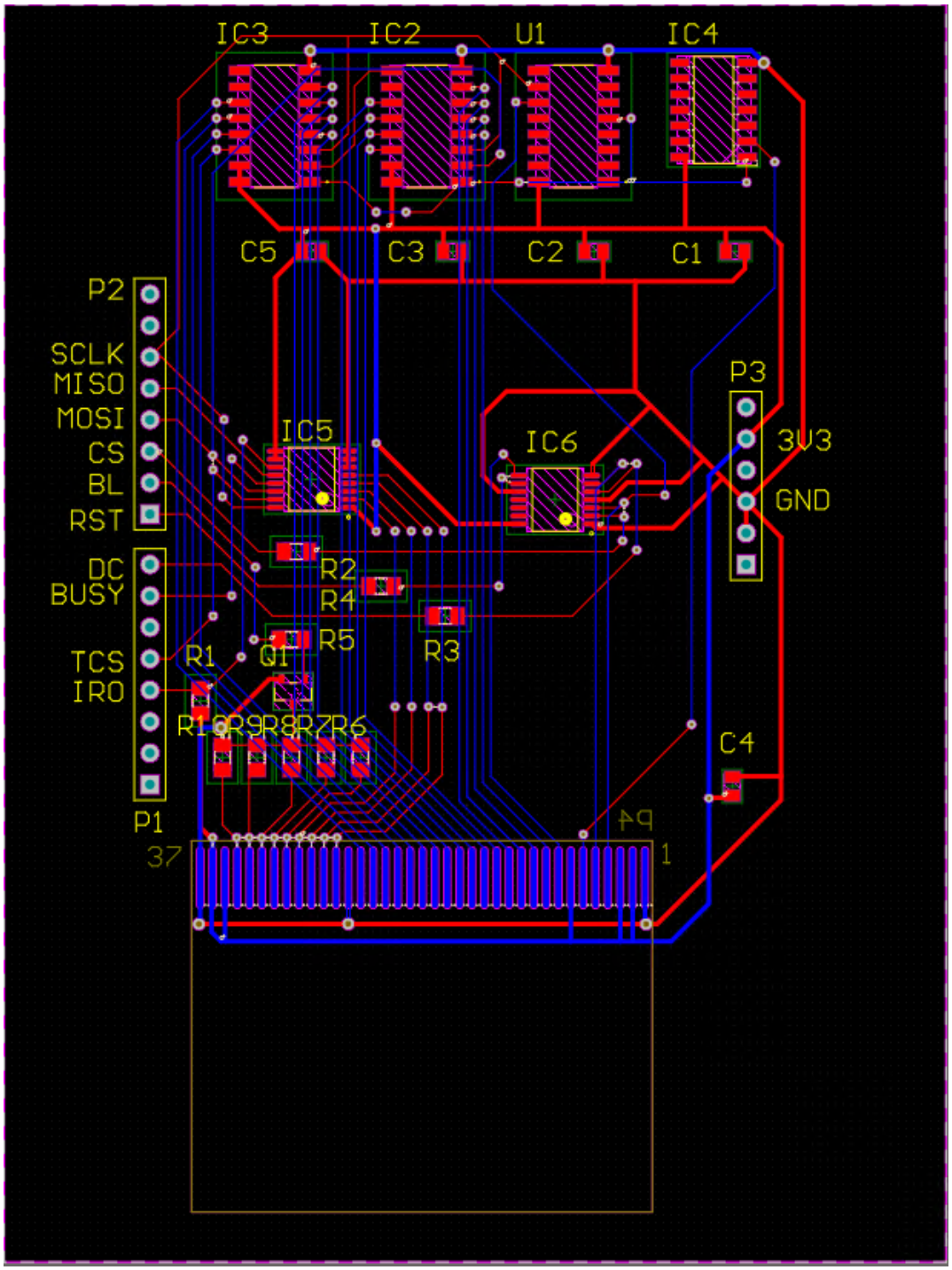

图 7 显示驱动模块 PCB 设计图

在 PCB 设计中我们根据 Mega2560 的 PCB 设计图，摆放了相应位置的 P1、P2、P3 插排。并且自定义了宽 0.6mm，长 5mm，引脚间距 1mm 的 37 引脚 FPC 裸板焊盘。

考虑到 FPC 折叠所需要的长度，我们最终设计了长 10.16cm，宽 7.62cm 的 PCB，使用 2 层板工艺制作。其中电容电阻使用 0805 封装，74HC4094 芯片以及 74HC4040 芯片采用 SOIC-16_150mil 封装，74HC04 芯片采用 SOIC-14_150mil 封装，XPT2046 芯片采用 TSSOP-16 封装，74VHC125 芯片采用 TSSOP-14 封装（由于采用 SOIC-14_150mil 的 74VHC125 芯片没有货，我们采用了更小的 TSSOP 封装）。

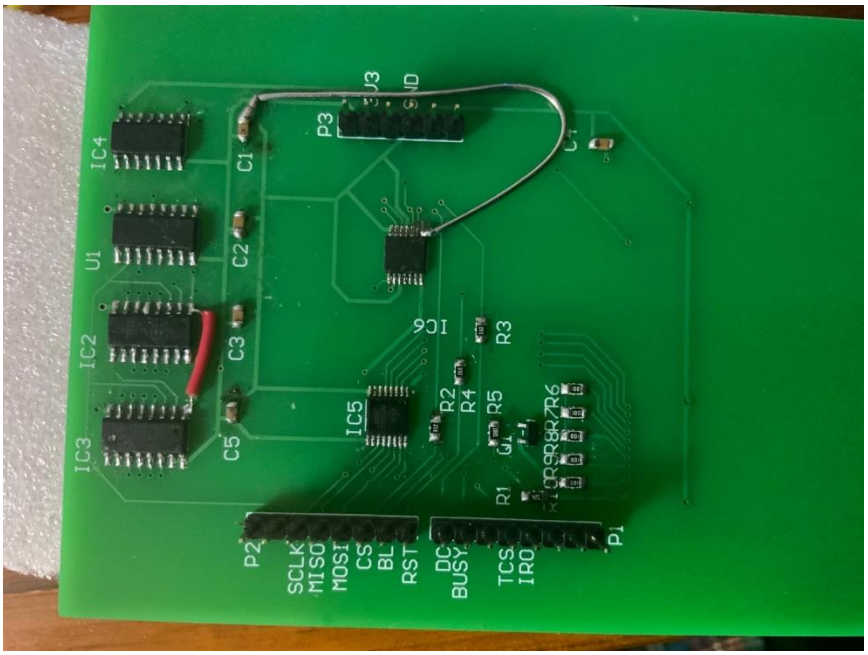

图 8 PCB 实物正面图

图 9 PCB 实物反面图

四、软件详细设计

4.1 软件系统结构

本系统的软件系统主要分为采样模块、数据处理模块、显示模块与触摸事件模块。

采样模块负责使用 10khz 的频率对待采总线进行采样。 数据处理模块负责对采样的信号进行同步以及解析。 显示模块负责系统界面的绘制以及采集信号与解析结果的可视化显示。触摸事件模块负责对显示屏的触摸信号进行监听，判断用户执行的各种操作，并形成事件以控制其它模块。系统采用异步调用形式进行集成。

系统的触发顺序可由时序图描述：

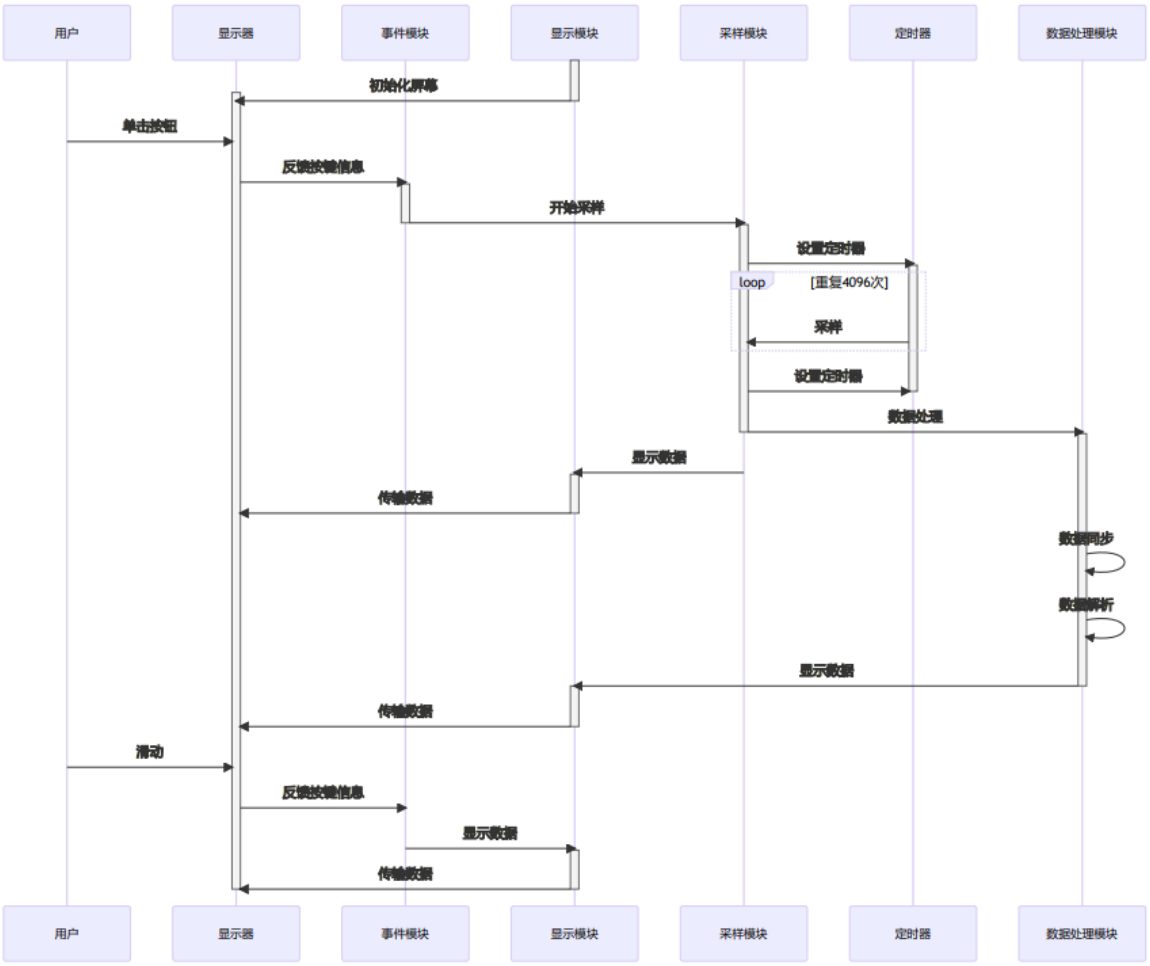

图 10 系统时序图

4.2 采集模块的软件设计

采集模块负责使用 10khz 的频率对待采总线进行采样，因此需要完成：

1. 计算采样长度
2. 初始化采样缓冲区
3. 设置定时器与中断

采样长度受限于系统内存，采样频率受限于采样长度与待测总线时钟频率。由于我们针对 1khz 的时钟进行采样，故我们使用 5 倍频的方式，即 10khz 进行采样。同时由于 mega2560 的内存限制与数组大小限制，我们选取 4096 长度的 char 数组进行数据存储。采样时间为 0.4096s，一个待采信号周期需要进行 10 次采样，因此在一次采样中，系统可以对 400 个时钟周期进行采样，获得 400 比特的数据，可以解析 50byte 的数据。

我们选取 Timer4 作为采样的定时器进行设置，在设置前我们需要计算预分频以及分配系数。

$$F = \frac{16MHz}{C_p * (C + 1)}$$

其中：

- F为定时器频率
- C_p为预分频系数
- C为分频系数

故我们选择如下参数进行配置，且配置 Timer4 的代码如图 4：

$$\begin{cases} C_p = 1 \\ C = 1599 \end{cases}$$

由于我们选取 Timer4 进行配置，我们还需要设置 Timer4 时钟中断函数。使用关键字 ISR 进行注册 TIMER4_COMPA_vect。

```
//设置Timer4
void SetTimer4(int x)
{
  //16,000,000/x hz
  noInterrupts();
  TCCR4A = 0;
  TCCR4B = 0;
  TCNT4 = 0;
  OCR4A = x;
  TCCR4B |= (1 << WGM42);
  // 设置1预分频
  TCCR4B |= (1 << CS10);
  TIMSK4 |= (1 << OCIE4A);
  interrupts();
}
```

图 11 10khz 时钟设置代码

在采样时，我们可以使用 digitalRead 进行采样，但是 digitalRead 实际是对端口寄存器的封装。

```
int digitalRead(uint8_t pin)
{
    uint8_t timer = digitalPinToTimer(pin);
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);

    if (port == NOT_A_PIN) return LOW;

    // If the pin that support PWM output, we need to turn it off
    // before getting a digital reading.
    if (timer != NOT_ON_TIMER) turnOffPWM(timer);

    if (*portInputRegister(port) & bit) return HIGH;
    return LOW;
}
```

图 12 digitalRead 源码

其在性能上会严重拉低 IO 速率，因此我们直接访问端口寄存器，由于我们需要采集 SCL，SDA 引脚，其均属于 PD 端口寄存器，且偏移量为 0,1，故我们可以使用 portInputRegister 宏函数进行采样，需要注意该函数返回的是寄存器指针，需要进行求值操作才可获得当前值。

由于采样时，我们并不知道信号从何时开始，由于我们的采样窗口非常小，因此我们需要从第一次信号发生变化处进行采集，也可以使用 I2C 相关的定义进行处理。我们使用第一种方式进行处理，因此我们设计了一个 DFA 进行检测，如图 12，在采样开始时，先设置定时器，再将 Mode 置为 0 即可开始采样。

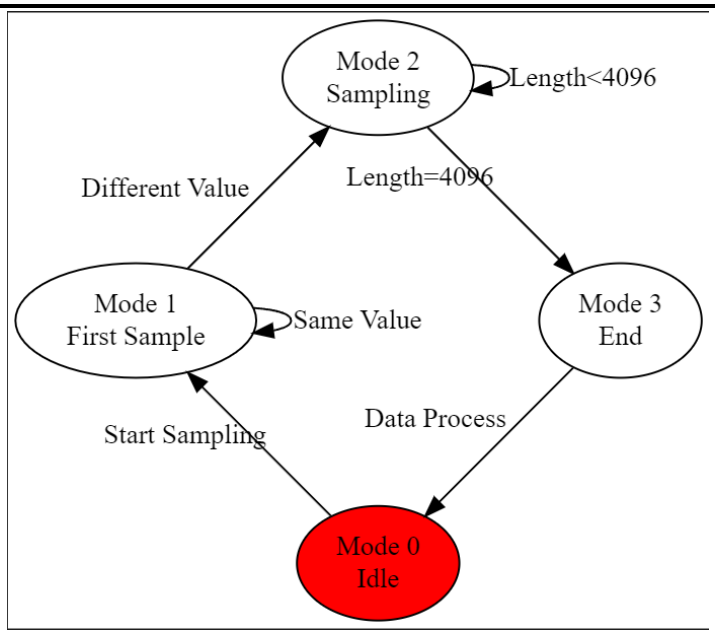

图 13 采样状态转移图

4.3 数据处理模块的软件设计

数据处理模块主要负责同步以及解析已采样数据。其中同步即同步 SCL 时钟，并将 SDA 解析为比特流，解析是指将比特流中传输的信息解析为数据流。

同步规则应对应 I2C 协议规则：

表 1 I2C 协议规则

事件	SCL	SDA
开始	高电平	下降沿
0	下降沿	低电平
1	下降沿	高电平
结束	高电平	上升沿

并且在 SCL 高电平时，SDA 应保持不变。

但是在实际使用当中，我们发现由于采样频率限制，会出现在 SCL 下降沿的同时，SDA 进行跳变。因此我们设计了相应的 DFA 进行同步（图 7）。在 Mode 2 与 Mode 4 向 Mode 3 或 Mode 5 转移时会写入比特 1 或 0。

在解析过程中，我们需要对比特流的合法性进行检测，I2C 在数据传输前，会发送数据头：

表 2 I2C Header

字段	Address	R/W	ACK
长度	7	1	1

在传输字节是会发送：

表 3 I2C Data

字段	Data	ACK
长度	8	1

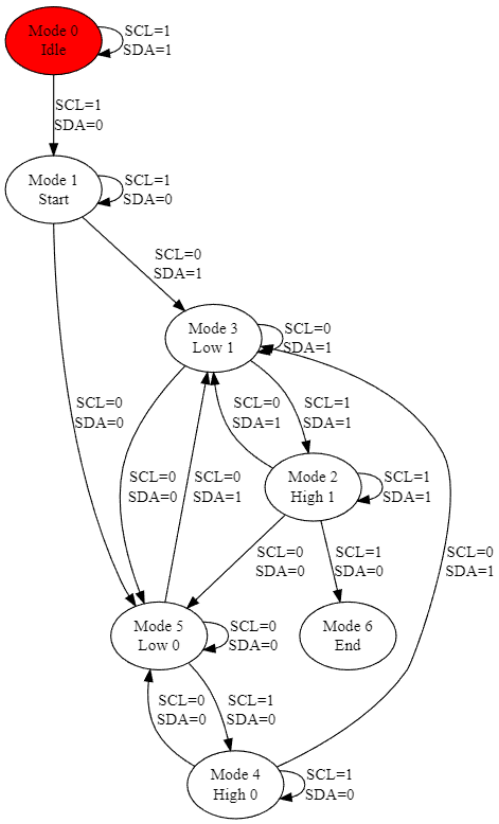

图 14 同步状态转移图

因此我们发现，合法的比特流长度应为 9 的整数倍，但是由于我们定义设计的 DFA 在

Mode 1 状态下设置比特 0，也为了方便我们读取字节更加规整，我们会在比特流开始前添加一位 0，因此我们合法的比特流长度为$9n + 10$。

4.4 触摸事件模块的软件设计

触摸事件模块主要负责定时读取电容屏幕的四边电压，据此判断用户是否触摸屏幕以及计算触点的坐标。并根据时间序列判定用户所执行的动作，以此触发相应的按钮。

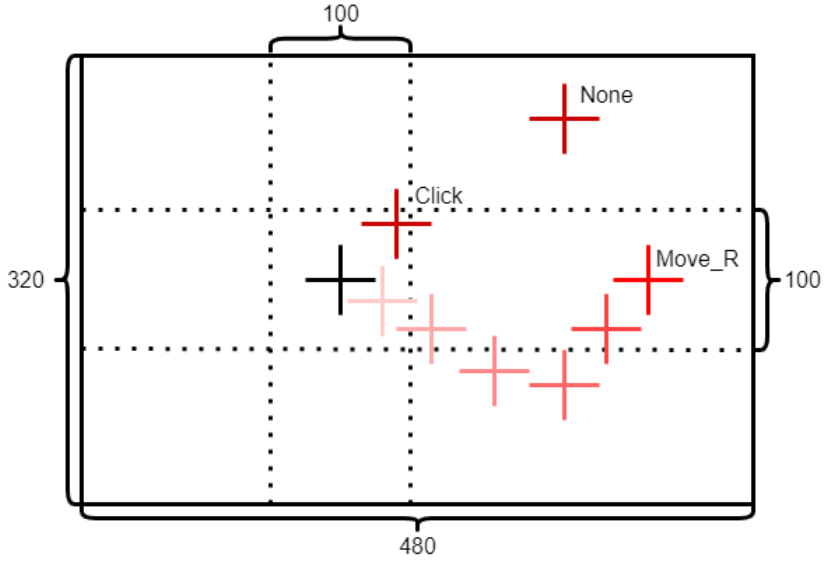

图 15 触摸事件判定

程序追踪用户开始触摸屏幕到触摸结束的触摸点坐标，依据图 14 中的规则进行判断，若起始与终止坐标在 50 以内，则判定为单击事件，若横纵坐标有一个变动在 50 以上，则判定移动事件，若均在 50 以上，则判定为非法事件，不予处理。

程序通过初始化按钮，来注册相关的单击按钮事件。

```
Button_Redraw = new Button(10, 5, 80, 20, "Redraw", Redraw, 8, 4);
Button_Adjust = new Button(380, 5, 80, 20, "Adjust", Adjust, 8, 4);
Button_I2C = new Button(160, 5, 40, 20, "I2C", I2C, 3, 4);
Button_Search = new Button(300, 5, 80, 20, "Search", Search, 8, 4);
Button_Add = new Button(180, 300, 20, 20, "+", AddCount, 5, 4);
Button_Reduce = new Button(20, 300, 20, 20, "-", ReduceCount, 5, 4);
Button_Line = new Button(200, 300, 60, 20, "Line", ChangeLine, 8, 4)
```

图 16 注册按钮代码

相应的，在模块中，我们在每次 Click 事件时，检测坐标是否在按钮内，命中时调用绑定的事件即可。表 1 中记录了系统允许的各种事件以及相关回调。

```
if (event->Type == Click)
{
  Button_Redraw->ClickTest(event->XPoint, event->YPoint);
  Button_Adjust->ClickTest(event->XPoint, event->YPoint);
  Button_I2C->ClickTest(event->XPoint, event->YPoint);
  Button_Search->ClickTest(event->XPoint, event->YPoint);
  Button_Add->ClickTest(event->XPoint, event->YPoint);
  Button_Reduce->ClickTest(event->XPoint, event->YPoint);
  Button_Line->ClickTest(event->XPoint, event->YPoint);
}
```

图 17 命中按钮检测代码

表 4 事件表

事件	功能	
向左滑动	使得波形图向左移动 Time 单位	
向右滑动	使得波形图向右移动 Time 单位	
向上滑动	查看前四个解析结果	
向下滑动	查看后四个解析结果	
单击	按钮	回调
	Redraw	重新绘制整个屏幕
	I2C	开始 I2C 采样
	Adjust	触摸屏校准
	Search	从当前位置搜索下一次传输
	+	增大显示数量（Time）
	-	减小显示数量（Time）
	Line	开关纵向辅助线显示

4.5 显示模块的软件设计

显示模块主要负责 GUI 的显示以及数据显示，GUI 主要包括按钮绘制，提示信息显示等，数据包括采样的波形图、解析结果图以及解析结果绘制。

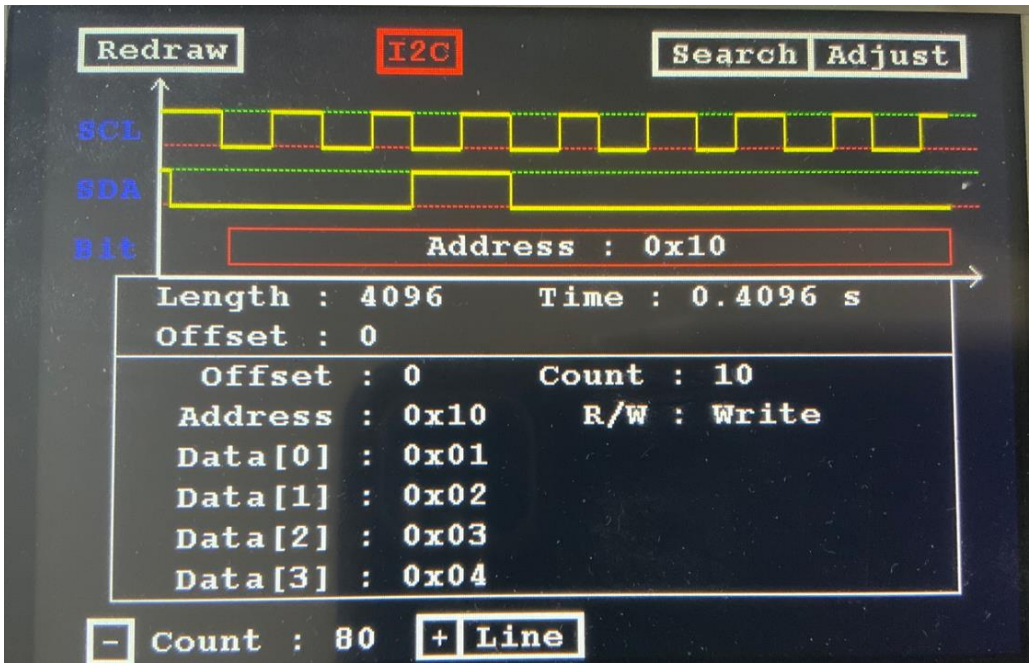

图 18 GUI

在绘制波形图时，为了便于观察细节与整体，我们对分辨率以及绘制长度进行了设置，考虑到屏幕的分辨率为 480*320，因此我们选择绘制的最大像素为 400px，这意味着：

$$Time * Width = 400$$

因此我们设置了七个档位：

表 5 绘制长度对应表

Level	0	1	2	3	4	5	6
Width	1	2	4	5	10	20	40
Time	400	200	100	80	40	20	10

```
//绘制数据帧
void PrintDataFrame(int channel, int offset, int mask, int length)
{
  if (offset > dataFrame->Length)
    offset = dataFrame->Length;
  if (offset + length > dataFrame->Length)
    length = dataFrame->Length - offset;
  if (length == 0)
    return;
  int h = 30 * channel + Offset_U + 16;
  int l = 30 * channel + Offset_U + 34;
  int b = (dataFrame->Data[offset] & mask) ? h : l;
  dataFrame->Peek(offset);
  for (int i = 0; i < length; i++)
  {
    int t = (dataFrame->Get() & mask) ? h : l;
    GUI_DrawRectangle(i * Width + Offset_L, t - 1, i * Width + Offset_L + Width, t + 1, YELLOW, DRAW_FULL, DOT_PIXEL_1X1);
    if (b != t)
    {
      if (b < t)
        GUI_DrawRectangle(i * Width + Offset_L, b, i * Width + Offset_L + 1, t, YELLOW, DRAW_FULL, DOT_PIXEL_1X1);
      else
        GUI_DrawRectangle(i * Width + Offset_L, t, i * Width + Offset_L + 1, b, YELLOW, DRAW_FULL, DOT_PIXEL_1X1);
      b = t;
    }
  }
}
```

图 19 波形图绘制

由于我们采样长度为 4096，而我们绘制的最大长度为 400，因此我们需要使用 Offset 进行滚动绘制，允许用户向左/右滑动屏幕以增大/减小 Offset，从而进行左右移动，偏移量的计算方法为：

$$Offset' = f(x) = \begin{cases} \max(0, Offset - Time), & Move\_L \\ \min\left(\max(0, 4096 - Time), Offset + Time\right), & Move\_R \end{cases}$$

对于解析数据，我们需要在波形图的下方进行绘制，因此我们需要在数据处理模块记录偏移量，这样我们就可以根据偏移量 Offset 以及绘制长度 Time 判断每个字节是否应当绘制。在确认绘制后，应当绘制相应的字符串，但是在绘制大小不足时，我们应当自动地更改显示字符串的长度，因此我们设置了三种表示方法：

表 6 显示字符串对应表

种类	原始	详细信息	粗略信息	简要信息
Address	Address=0x10	Address : 0x10	0x10	10
Data	Data[1]=0x1A	Data[1] : 0x1A	0x1A	1A

```
//打印地址
void PrintAddress()
{
  if(Result->End<Offset)return;
  if(Result->Start>Offset+Time)return;
  int start=max(Offset,Result->Start)-Offset;
  int end=min(Offset+Time,Result->End)-Offset;

  //进行嗅探操作，找到大小不超过绘制长度的合法显示
  sprintf(s,"Address : 0x%c%c",transcode[Result->Address>>4],transcode[Result->Address&15]);

  if((end-start)*Width>strlen(s)*12)
  {
    int o=((end-start)*Width-strlen(s)*12)/2;
    GUI_DisString_EN(Offset_L+start*Width+o, 30 * 2 + Offset_U + 18, s, &Font16, LCD_BACKGROUND, WHITE);
  }
  else
  {
    sprintf(s,"0x%c%c",transcode[Result->Address>>4],transcode[Result->Address&15]);
    if((end-start)*Width>strlen(s)*12)
    {
      int o=((end-start)*Width-strlen(s)*12)/2;
      GUI_DisString_EN(Offset_L+start*Width+o, 30 * 2 + Offset_U + 18, s, &Font16, LCD_BACKGROUND, WHITE);
    }
    else
    {
      sprintf(s,"%c%c",transcode[Result->Address>>4],transcode[Result->Address&15]);
      if((end-start)*Width>strlen(s)*12)
      {
        int o=((end-start)*Width-strlen(s)*12)/2;
        GUI_DisString_EN(Offset_L+start*Width+o, 30 * 2 + Offset_U + 18, s, &Font16, LCD_BACKGROUND, WHITE);
      }
    }
  }

  GUI_DrawRectangle(Offset_L+start*Width, 30 * 2 + Offset_U + 16, Offset_L + end * Width,
                    30 * 2 + Offset_U + 34, RED, DRAW_EMPTY, DOT_PIXEL_1X1);
}
```

图 20 打印地址代码

五、系统测试

我们搭建了一个测试环境，使用 I2C 转 USB 调试仪向 Arduino 开发板发送指定数据，并使用我们设计的系统进行采样与分析，通过比对系统显示在屏幕上的解析数据与发送的数据，即可验证系统的正确性。

1. 初始化系统

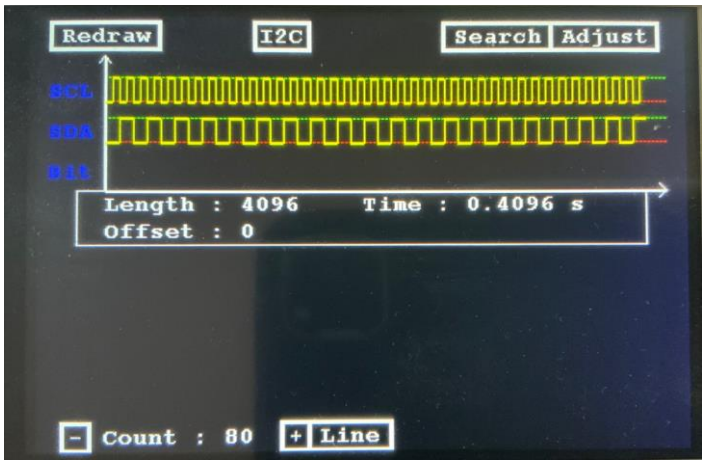

图 21 初始化界面

可以看到，系统可以正确地显示初始化界面，包括 7 个按钮、初始化波形图、采样信息以及绘制长度。

2. 开启 I2C 采样

单击 I2C 按钮，系统进入采样阶段，屏幕右下角会出现闪烁的 Sampling 标识。

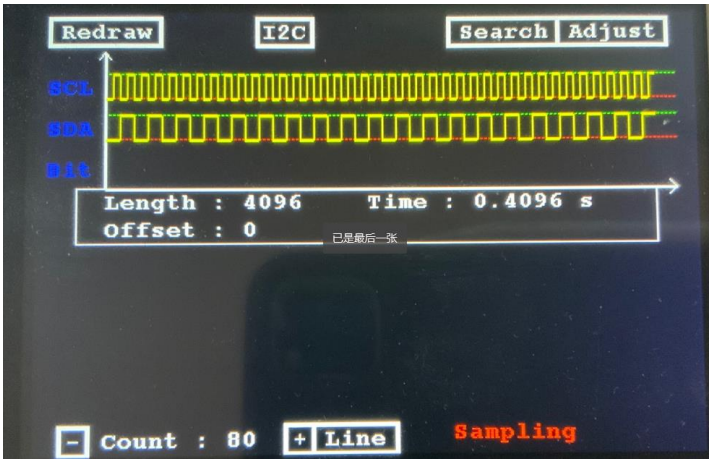

图 22 采样界面

3. 使用调试仪发送数据

我们使用调试仪作为 I2C Master 向 Arduino 中的 I2C Slaver 发送指定数据，其中 I2C Slaver 的地址为 0x10，发送两帧数据：

- 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
- 0x02

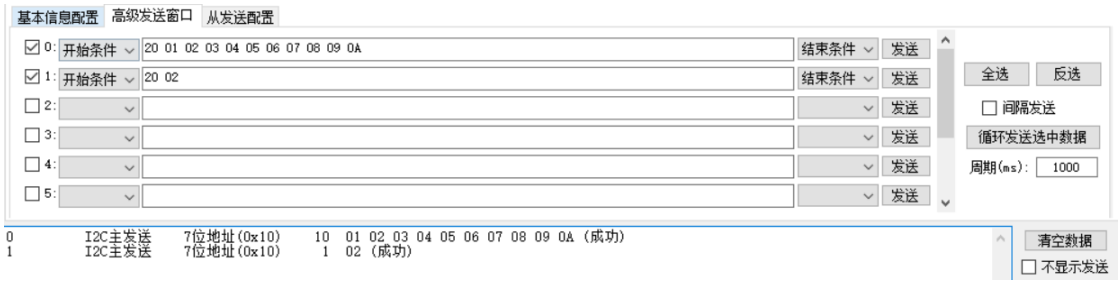

图 23 I2C Master

数据发送成功后会显示发送成功，在发送失败时应检查虚拟端口是否开启。

4. 采样成功

在发送成功后，系统会从采样界面转跳到采样成功界面，在界面中显示采样到的波形图。

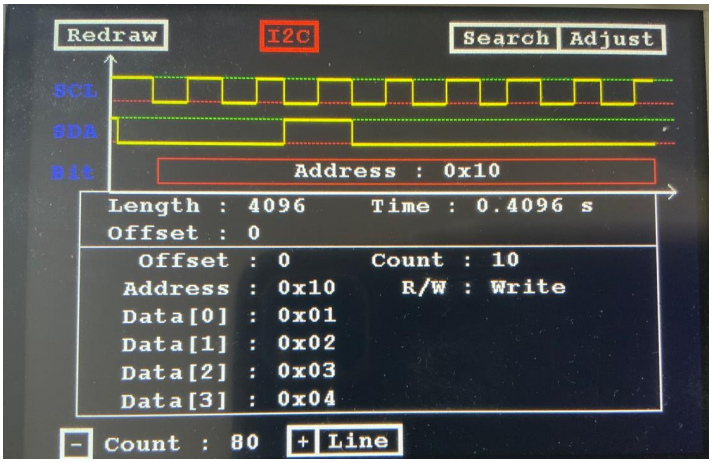

图 24 采样成功界面

采样完成后，I2C 按钮会变红，且 SCL 与 SDA 变成采样信号，在 BIT 行，显示红色 Address 框，其中显示详细信息，注意到 Address 与 Slaver 的地址相同。

此时在解析信息栏中出现了解析的详细信息，包括起始偏移量为 0，这是由于我们采样是检测第一次跳变，即 I2C 的开始信号，因此第一帧的偏移量为 0。解析地址为 0x10 与预期相符，RW 为 Write 与预期相符，数据长度为 10，前四个字节数据为 0x01 0x02 0x03 0x04，与预期相符。

5. 向右滑动

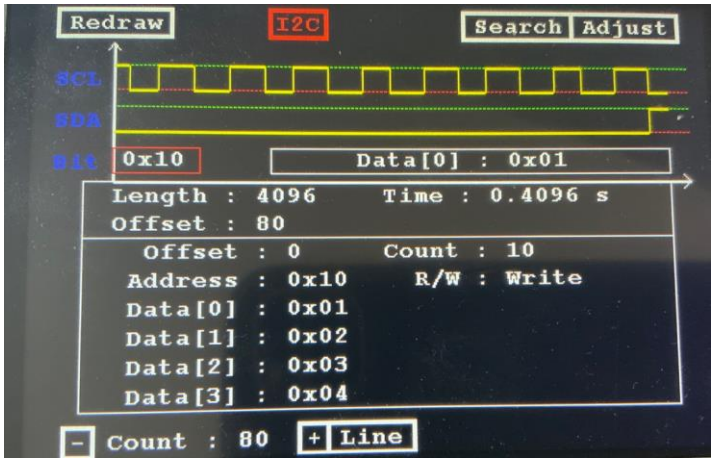

图 25 向右滑动

向右滑动屏幕后，Offset 变为 80，增加量与 Time 80 相等，SCL、SDA、BIT 重新绘制为偏移量 80 的波形图，并且 BIT 显示第一位字节的详细信息，由于 Address 长度不够，

故显示 Address 的缩略信息，0x10。

6. 向下滑动

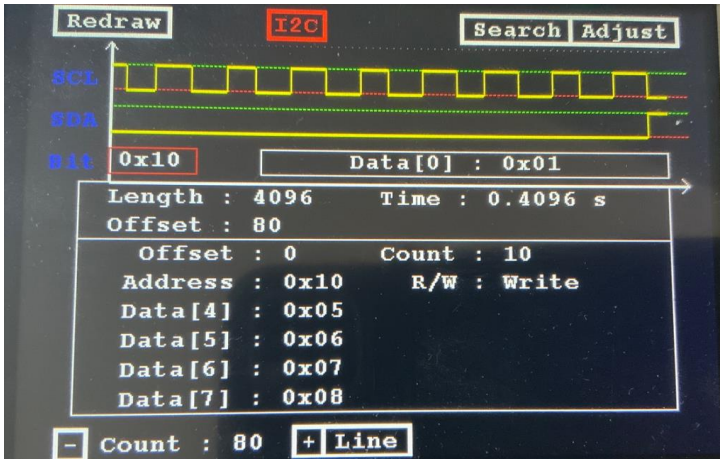

图 26 向下滑动

向下滑动屏幕后，显示第五位开始到第八位解析字节为 0x05 0x06 0x07 0x08。

7. 缩放

点击+，进行缩放。缩放后，绘制长度发生改变，并且绘制的解析字节长度不足，故只显示缩略字符串。

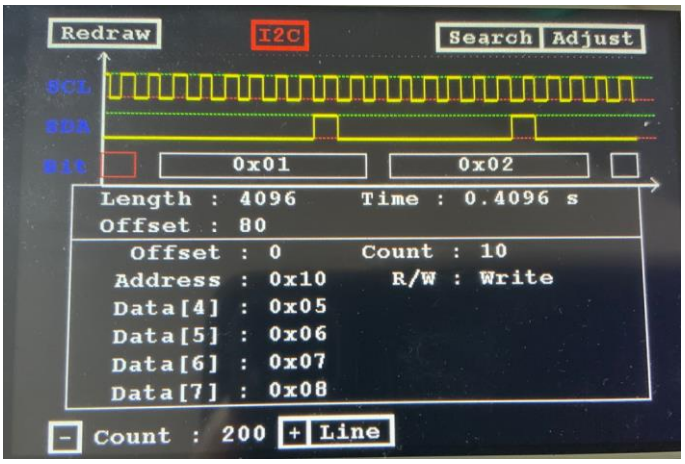

图 27 缩放结果

8. 显示纵向辅助线

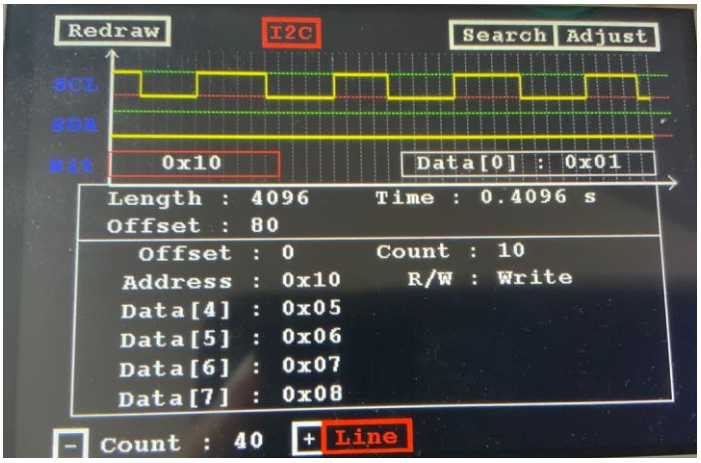

图 28 显示纵向辅助线

点击-号按钮并单击 Line 按钮，显示纵向辅助线（由于在绘制长度为 200 时开启纵向辅助线会导致屏幕太过混乱，绘制时间过长，因此我们选择先缩放再测试）。单击后，可

以看到 Line 按钮变红，且波形图中出现纵向的虚线辅助线。

9. I2C 搜索

单击 Search 按钮后，我们发现 Offset 发生改变，从 0 变成了 958，并且波形图也跟着发生了偏移，此时解析的结果也进行了重新绘制，Address 依然为 0x10 与预期相符，发送的字节数量为 1，介绍到的字节数据为 0x02，与预期相符。

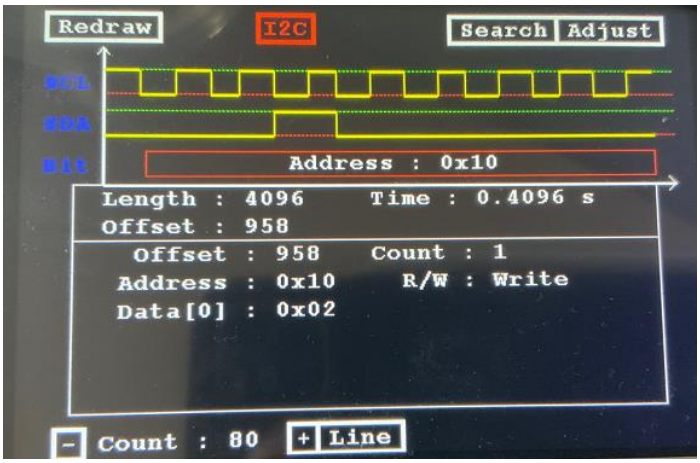

图 29 搜索完成

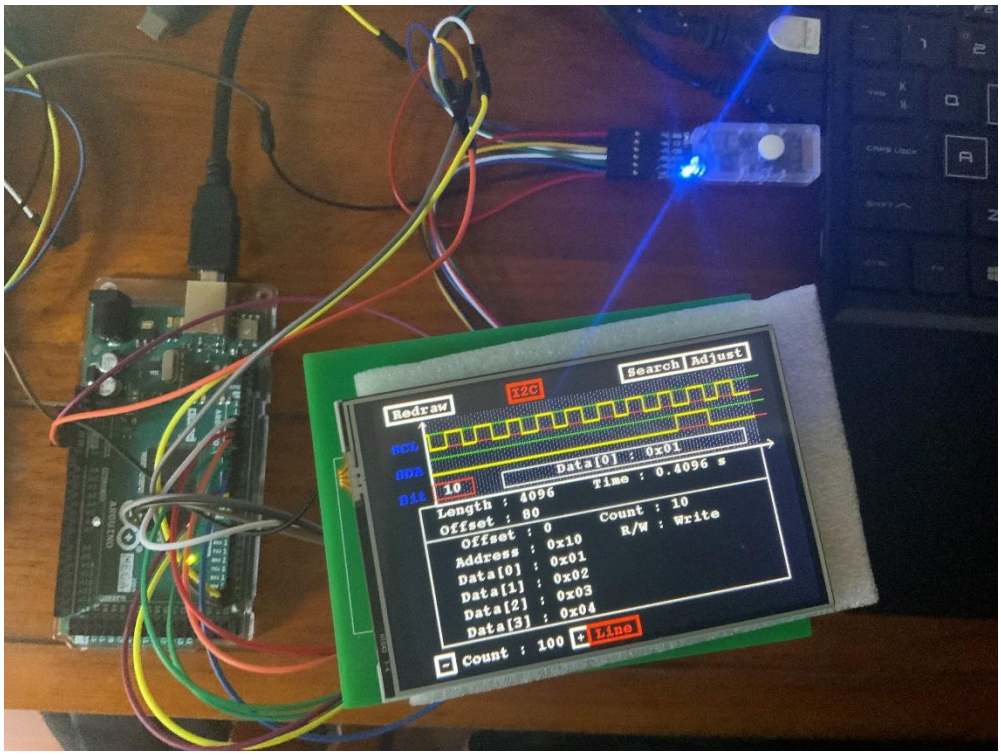

图 30 实物演示图

六、总结

由于疫情的原因，计算机硬件课设陪伴了我们将近一年的时间，从大三下到大四上，这让我们有更多的时间思考我们为什么要学计算机科学与技术。计算机不仅仅是一个专业，更是我们的一种工具，它能帮助我们解决生活中遇到的各种问题，计算机思维，系统思维，自顶向下与自底向上的分层思维是让我们理性思考问题、解决问题的重要手段。不仅是软件，能帮助我们编写代码，实现自动化作业，硬件也是我们生活中不可或缺的一部分，生活离不开硬件，软硬件结合更是最贴近生活的解决方案。硬件课设这门课给予了我们一次使用计算机思维，应用软硬件结合的方式，解决生活中问题的机会。在这门课程中，我们发现问题，提出问题，解决问题，并且不断提高自身，找到自身的不足，尽力做到最好。

不开玩笑的说，这是我第二次与硬件打交道，第一次还是在高中，我只负责处理硬件传送回来的数据，而这次我们不仅要使用硬件，还用设计硬件，制造硬件，这对我来说是一次挑战。

在课程中，我发现自身的最大问题就是风险意识薄弱，虽然我们有周志，时刻记录着计划与进度，但是计划赶不上变化，芯片的供需关系导致的芯片短缺打得我们一个措手不及，不得不修改封装，重新制版。Mega2560 的理论与实际的 IO 速率，让我们草率地制定了设计方案却不得不向现实妥协，转而使用更加昂贵的 I2C 转 USB 调试器方案。眼前的一点可能让我们兴奋地迷失了方向，不知更大地风险正在前方等着我们。风险管理，往大了说关乎企业的兴衰与存亡，往小了说关乎一个项目的时间成本与金钱成本，这是我还需要学习的功课。

附录

参考资料：

[1] Mega2560 数据手册
https://docs.arduino.cc/static/0b37f2ff5f702c1e7cfb643fcc741ce8/A000067-datasheet.pdf
[2] Mega2560 PCB 文件
https://docs.arduino.cc/static/d1927c3a6f7cce0a944fb502ae402e20/eagle-files.zip
[3] TFT 触摸屏官方文档 https://www.waveshare.net/wiki/4inch_TFT_Touch_Shield
[4] I2C 参考文档 https://www.arduino.cc/reference/en/libraries/i2c/

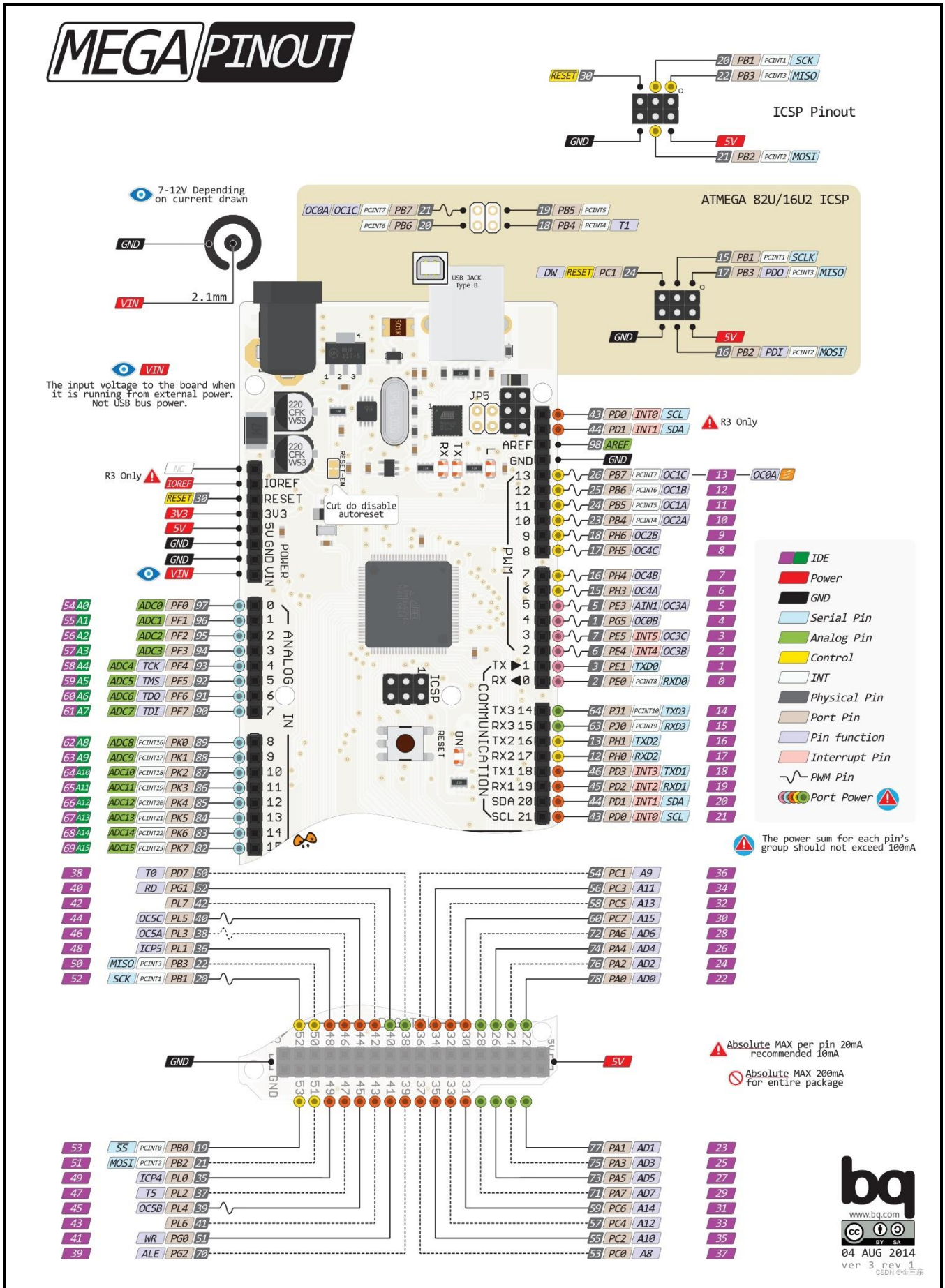

图 31 Mega2560 引脚图

Program.ino 源码

```
#include "DEV_Config.h"
#include "LCD_Driver.h"
#include "LCD_GUI.h"
#include "LCD_Touch.h"
#include <Wire.h>
#include "Data.hpp"
#include "Button.hpp"

//数据帧
DataFrame_I2C *dataFrame;
//采样信息
SampleInfo *sampleInfo;

//转义字符
char transcode[]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
//sprintf 临时字符串
char *s=new char[128];
//dtostrf 临时字符串（Arduino 中的 sprintf 不支持 double）
char *ts=new char[16];

//GUI 偏移量
#define Offset_L 50
#define Offset_U 30

//绘制间隔
#define WidthCount 6
int Widths[]{1,2,4,5,10,20,40};
int WIndex=3;
int Width=5;
int Time=80;
//是否绘制纵向辅助线
bool Column=false;

//按钮
Button *Button_Redraw;
Button *Button_Adjust;
Button *Button_I2C;
Button *Button_Search;
Button *Button_Add;
Button *Button_Reduce;
Button *Button_Line;

char* Names[3]{"SCL","SDA","Bit"};
```

```
//采样数据绘制起始地址
int Offset = 0;

//解析结果绘制起始地址
int DOffset=0;

//设置 Timer4
void SetTimer4(int x)
{
  //16,000,000/x hz
  noInterrupts();
  TCCR4A = 0;
  TCCR4B = 0;
  TCNT4 = 0;
  OCR4A = x;
  TCCR4B |= (1 << WGM42);
  // 设置 1 预分频
  TCCR4B |= (1 << CS10);
  TIMSK4 |= (1 << OCIE4A);
  interrupts();
}

//设置 Timer1
void SetTimer()
{
  //1hz
  noInterrupts();
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 0;
  OCR1A = 15624;
  TCCR1B |= (1 << WGM12);
  // 设置 1024 预分频
  TCCR1B |= (1 << CS12) | (1 << CS10);
  TIMSK1 |= (1 << OCIE1A);
  interrupts();
}

int Mode;

// 绘制采样标识
int _COLOR=0;

void ClearTip()
{
```

```
  GUI_DrawRectangle(300,300,400,320,BLACK,DRAW_FULL,DOT_PIXEL_1X1);
}
void SetTip(int color)
{
  GUI_DisString_EN(300,300, "Sampling", &Font16, BLACK, color);
}

//1hz 定时器，若在采样中，定时闪烁采样标识
ISR(TIMER1_COMPA_vect)
{
  if(Mode!=0)
  {
    if(_COLOR==1)
    {
      _COLOR=0;
      SetTip(BLACK);
    }
    else
    {
      _COLOR=1;
      SetTip(RED);
    }
  }
}

//10khz 定时器，对信号进行采样
int last;

ISR(TIMER4_COMPA_vect)
{
  //针对 SCL,SDA 针脚进行采样
  int x=*portInputRegister(4)&3;
  if(Mode==1)
  {
    last=x;
    Mode=2;
  }
  else if(Mode==2)
  {
    //若信号发生改变，则进行采样
    if(x!=last)
    {
      Mode=3;
      Serial.println(last);
      Serial.println(x);
```

```
      dataFrame->Peek(0);
      dataFrame->Set(last);
      dataFrame->Set(x);
    }
  }
  else if(Mode==3)
  {
    dataFrame->Set(x);
    //采样完成后，进行后续处理
    if(dataFrame->P>=dataFrame->End){
      Mode=0;
      Offset=0;
      dataFrame->Peek(0);
      while(dataFrame->P<dataFrame->End)
        Serial.println((int)dataFrame->Get());
      SetTimer4(1599);
      Button_I2C->FrontColor=RED;
      DrawButton(Button_I2C);
      Parse();
      ClearTip();
      DrawChannels();
    }
  }
}

//同步后的比特数据
bool Bits[512];
//比特对应的下降沿位置
int Indexs[512];
int BIndex=0;

//状态机
int mode=0;
void SetMode(int x,int next)
{
  //串口调试
  Serial.print(mode);
  Serial.print(",");
  Serial.print(x&1);
  Serial.print(",");
  Serial.print(x&2);
  Serial.print(",");
  Serial.println(next);
  mode=next;
}
```

```
//使用状态机进行同步
void Parse()
{
  BIndex=0;
  mode=1;
  dataFrame->Peek(Offset);
  while(dataFrame->P<dataFrame->End)
  {
    int x=dataFrame->Get();
    if(mode==1)
    {
      if(x==1)SetMode(x,2);
    }
    else if(mode==2)
    {
      if(x==0)
      {
        SetMode(x,3);
        Bits[BIndex]=0;
        Indexs[BIndex++]=dataFrame->P-dataFrame->Data;
        Serial.println(0);
      }
      else if(x==2)
      {
        SetMode(x,5);
        Bits[BIndex]=0;
        Indexs[BIndex++]=dataFrame->P-dataFrame->Data;
        Serial.println(0);
      }
      else if(x==3)
      {
        SetMode(x,1);
        Serial.println("STOP");
        ParseBit();
        return;
      }
    }
    else if(mode==3)
    {
      if(x==1)SetMode(x,2);
      else if(x==2)SetMode(x,5);
    }
    else if(mode==4)
    {
```

```
      if(x==2)
      {
        SetMode(x,5);
        Bits[BIndex]=1;
        Indexs[BIndex++]=dataFrame->P-dataFrame->Data;
        Serial.println(1);
      }
      else if(x==0)
      {
        SetMode(x,3);
        Bits[BIndex]=1;
        Indexs[BIndex++]=dataFrame->P-dataFrame->Data;
        Serial.println(1);
      }
    }
    else if(mode=5)
    {
      if(x==3)SetMode(x,4);
      else if(x==0)SetMode(x,3);
    }
  }
}

//解析结果
bool Error;
Package *Result;

unsigned char Read(int offset)
{
  unsigned char r=0;
  bool *p=Bits+offset;
  for(int i=7;i>=0;i--,p++)
    if(*p)
      r|=(1<<i);
  return r;
}

//对比特流进行解析
void ParseBit()
{
  //比特信息长度必须为9n+10
  if(BIndex%9!=1)
  {
    Serial.println("Error");
    Error=true;
```

```
    return;
  }
  Error=false;
  char address=Read(0);
  Serial.print("Address=");
  Serial.print("0x");
  Serial.print(transcode[address>>4]);
  Serial.println(transcode[address&15]);
  Serial.print("R/W=");
  bool rw=Bits[8];
  Serial.println(rw?"R":"W");
  bool ack=Bits[9];
  Serial.print("ACK=");
  Serial.println(ack?"NACK":"ACK");
  Serial.print("Count=");
  int count=(BIndex-10)/9;
  Serial.println(count);
  Result=new Package(address,rw,ack,count,Indexs[0],Indexs[9]);
  for(int i=0;i<count;i++)
  {
    Data *data=Result->Values+i;
    data->Value=Read(i*9+10);
    Serial.print("Data[");
    Serial.print(i);
    Serial.print("]=0x");
    Serial.print(transcode[data->Value>>4]);
    Serial.println(transcode[data->Value&15]);
    Serial.println((int)data->Value);
    data->ACK=Bits[i*9+10+8];
    Serial.print("ACK=");
    Serial.println(data->ACK?"NACK":"ACK");
    data->Start=Indexs[i*9+10];
    data->End=Indexs[i*9+10+8];
  }
  PrintResult();
}

//初始化
void setup()
{

  //开启 I2C Slaver
  Wire.begin(0x10);
  Wire.onReceive(receive);
  Serial.begin(921600);
```

```
  //LCD 初始化
  System_Init();
  SetTimer();
  SetTimer4(1599);
  sampleInfo = GetSampleInfo(sizeof(char), 4096);
  dataFrame = new DataFrame_I2C(sampleInfo->Length);
  dataFrame->Random();

  //触摸屏初始化
  LCD_SCAN_DIR Lcd_ScanDir = SCAN_DIR_DFT;
  LCD_Init(Lcd_ScanDir, 100);
  TP_Init(Lcd_ScanDir);
  TP_GetAdFac();

  Button_Redraw = new Button(10, 5, 80, 20, "Redraw", Redraw, 8, 4);
  Button_Adjust = new Button(380, 5, 80, 20, "Adjust", Adjust, 8, 4);
  Button_I2C = new Button(160, 5, 40, 20, "I2C", I2C, 3, 4);
  Button_Search = new Button(300, 5, 80, 20, "Search", Search, 8, 4);
  Button_Add = new Button(180, 300, 20, 20, "+", AddCount, 5, 4);
  Button_Reduce = new Button(20, 300, 20, 20, "-", ReduceCount, 5, 4);
  Button_Line = new Button(200, 300, 60, 20, "Line", ChangeLine, 8, 4);

  // GUI_DrawLine(0, 10, LCD_WIDTH, 10, RED, LINE_SOLID, DOT_PIXEL_2X2);

  // GUI_DrawRectangle(0, Offset_L, LCD_WIDTH, Offset_L+1, RED, DRAW_FULL,
DOT_PIXEL_2X2);

  LCD_Clear(BLACK);

  //绘制 GUI
  DrawChannels();
  PrintInfo();
  DrawButtons();
  DrawCount();

  //TP_Adjust();
}

//I2C Slaver
void receive(int len)
{
  Wire.read();
}
```

```
//Main Loop 处理触摸事件
void loop()
{
  if(Mode!=0)
  {
    delay(1);
    return;
  }
  //扫描按键事件
  TP_EVENT *event = Scan();
  if (event != nullptr)
  {
    event->Lock();
    // GUI_DrawRectangle(0,Offset_L0,300,300,RED,DRAW_FULL,DOT_PIXEL_1X1);
    // GUI_DisNum(0,Offset_L0,x,&Font16,RED,BLACK);
    // GUI_DisNum(0,Offset_U+50,y,&Font16,RED,BLACK);
    Serial.print(event->Start);
    Serial.print(",");
    Serial.print(event->XPoint0);
    Serial.print(",");
    Serial.print(event->YPoint0);
    Serial.print(",");
    Serial.print(event->End);
    Serial.print(",");
    Serial.print(event->XPoint);
    Serial.print(",");
    Serial.print(event->YPoint);
    Serial.print(",");
    Serial.print(event->ScanCount);
    Serial.print(",");
    if (event->Type == None)
      Serial.println("NONE");
    else if (event->Type == Click)
      Serial.println("Click");
    else if (event->Type == Move_L)
      Serial.println("Move_L");
    else if (event->Type == Move_R)
      Serial.println("Move_R");
    else if (event->Type == Move_D)
      Serial.println("Move_D");
    else if (event->Type == Move_U)
      Serial.println("Move_U");
    if (event->Type == Click)
    {
      Button_Redraw->ClickTest(event->XPoint, event->YPoint);
```

```
      Button_Adjust->ClickTest(event->XPoint, event->YPoint);
      Button_I2C->ClickTest(event->XPoint, event->YPoint);
      Button_Search->ClickTest(event->XPoint, event->YPoint);
      Button_Add->ClickTest(event->XPoint, event->YPoint);
      Button_Reduce->ClickTest(event->XPoint, event->YPoint);
      Button_Line->ClickTest(event->XPoint, event->YPoint);
    }
    else if (event->Type == Move_R)
    {
      //向右滑动，Offset 增加 Time
      Offset = min(Offset+Time,sampleInfo->Length-Time);
      DrawChannels();
    }
    else if (event->Type == Move_L)
    {
      //想做滑动 Offset 减少 Time
      Offset = max(0, Offset - Time);
      DrawChannels();
    }
    else if(event->Type == Move_U)
    {
      //向上滑动，且解析成功后
      if(Result!=nullptr)
      {
        //解析数据向上滑动 4
        DOffset=max(0,DOffset-4);
        PrintDatas();
      }
    }
    else if(event->Type == Move_D)
    {
      //向下滑动，且解析成功后
      if(Result!=nullptr)
      {
        //解析数据向下滑动 4
        DOffset=min(max(0,Result->Count-4),DOffset+4);
        PrintDatas();
      }
    }
  }
  delay(10);
}

//绘制坐标系
void PrintChannelLines(int count)
```

```
{

  int h = count * 30 + 40;

  int r = Time * Width + 10;

  GUI_DrawRectangle(Offset_L,Offset_U,r+Offset_L,h+Offset_U,LCD_BACKGROUND,DRAW_FUL
L,DOT_PIXEL_1X1);

  if(Column)
    for (int i = 0; i < Time; i++)
      GUI_DrawLine(i * Width + Offset_L + Width, Offset_U, i * Width + Offset_L +
Width, h + Offset_U, GRAY, LINE_DOTTED, DOT_PIXEL_1X1);

  for (int i = 0; i < count; i++)
  {
    GUI_DrawLine(Offset_L, 30 * i + Offset_U + 16, Offset_L + r + 5, 30 * i +
Offset_U + 16, GREEN, LINE_DOTTED, DOT_PIXEL_1X1);
    GUI_DrawLine(Offset_L, 30 * i + Offset_U + 34, Offset_L + r + 5, 30 * i +
Offset_U + 34, RED, LINE_DOTTED, DOT_PIXEL_1X1);
    //GUI_DisNum(9, 30 * i + Offset_U + 20, i, &Font16, LCD_BACKGROUND, BLUE);
    GUI_DisString_EN(9,30*i+Offset_U+20,Names[i],&Font16,LCD_BACKGROUND, BLUE);
  }

  GUI_DisString_EN(9,60+Offset_U+20,Names[2],&Font16,LCD_BACKGROUND, BLUE);

  GUI_DrawRectangle(Offset_L - 1, Offset_U, Offset_L + 1, h + Offset_U, WHITE,
DRAW_FULL, DOT_PIXEL_2X2);

  GUI_DrawLine(Offset_L - 4, Offset_U + 5, Offset_L, Offset_U, WHITE, LINE_SOLID,
DOT_PIXEL_1X1);

  GUI_DrawLine(Offset_L + 4, Offset_U + 5, Offset_L, Offset_U, WHITE, LINE_SOLID,
DOT_PIXEL_1X1);

  GUI_DrawRectangle(Offset_L, h + Offset_U - 1, Offset_L + r + 5, h + Offset_U + 1,
WHITE, DRAW_FULL, DOT_PIXEL_2X2);

  GUI_DrawLine(Offset_L + r, h + Offset_U + 5, Offset_L + r + 5, h + Offset_U,
WHITE, LINE_SOLID, DOT_PIXEL_1X1);

  GUI_DrawLine(Offset_L + r, h + Offset_U - 5, Offset_L + r + 5, h + Offset_U,
WHITE, LINE_SOLID, DOT_PIXEL_1X1);

  DrawOffset();
```

```
}

//打印偏移量
void DrawOffset()
{
  GUI_DrawRectangle(Offset_L, 70 + 60 + 24, 400, 70 + 60 + 36, BLACK, DRAW_FULL,
DOT_PIXEL_1X1);

  sprintf(s, "Offset : %d", Offset);

  GUI_DisString_EN(Offset_L, 70 + 60 + 20 + 4, s, &Font16, BLACK, WHITE);
}

//打印采样信息
void PrintInfo()
{
  int infoh = 70 + 60;

  GUI_DrawRectangle(Offset_L-20, infoh, 450, infoh + 40, WHITE, DRAW_EMPTY,
DOT_PIXEL_1X1);

  sprintf(s, "Length : %ld", sampleInfo->Length);

  GUI_DisString_EN(Offset_L, infoh + 4, s, &Font16, BLACK, WHITE);

  dtostrf(sampleInfo->Time, 6, 4, ts);

  sprintf(s, "Time : %s s", ts);

  GUI_DisString_EN(LCD_WIDTH/2, infoh + 4, s, &Font16, BLACK, WHITE);
}

//打印数据单元
void PrintDatas()
{
  GUI_DrawRectangle(Offset_L, 70 + 140, 400, 70 + 140 + 76, BLACK, DRAW_FULL,
DOT_PIXEL_1X1);
  for(int i=0;i<4;i++)
  {
    int offset=DOffset+i;
    if(0<=offset&&offset<Result->Count)
    {
      Serial.print("PrintData:");
      Serial.print(offset);
      Serial.print(",");
```

```
      Serial.println(Result->Values[offset].Value);
      int h=70+140+i*20+4;
      sprintf(s, " Data[%d] : 0x%c%c", offset,
transcode[Result->Values[offset].Value>>4],transcode[Result->Values[offset].Value&1
5]);
      GUI_DisString_EN(Offset_L, h, s, &Font16, BLACK, WHITE);
    }
  }
}

//打印数据包
void PrintResult()
{
  int infoh = 70 + 100;

  GUI_DrawRectangle(Offset_L-20, infoh, 450, infoh + 120, BLACK, DRAW_FULL,
DOT_PIXEL_1X1);

  GUI_DrawRectangle(Offset_L-20, infoh, 450, infoh + 120, WHITE, DRAW_EMPTY,
DOT_PIXEL_1X1);

  sprintf(s, "  Offset : %d", Offset);

  GUI_DisString_EN(Offset_L, infoh + 4, s, &Font16, BLACK, WHITE);

  sprintf(s, " Address : 0x%c%c",
transcode[Result->Address>>4],transcode[Result->Address&15]);

  GUI_DisString_EN(Offset_L, infoh + 24, s, &Font16, BLACK, WHITE);

  sprintf(s, "Count : %d", Result->Count);

  GUI_DisString_EN(LCD_WIDTH/2, infoh + 4, s, &Font16, BLACK, WHITE);

  sprintf(s, "  R/W : %s", Result->RW?"Read":"Write");

  GUI_DisString_EN(LCD_WIDTH/2, infoh + 24, s, &Font16, BLACK, WHITE);

  DOffset=0;

  PrintDatas();
}

//绘制按钮
void DrawButtons()
```

```
{
  DrawButton(Button_Redraw);
  DrawButton(Button_Adjust);
  DrawButton(Button_I2C);
  DrawButton(Button_Search);
  DrawButton(Button_Add);
  DrawButton(Button_Reduce);
  DrawButton(Button_Line);
}

//绘制数据帧
void PrintDataFrame(int channel, int offset, int mask, int length)
{
  if (offset > dataFrame->Length)
    offset = dataFrame->Length;
  if (offset + length > dataFrame->Length)
    length = dataFrame->Length - offset;
  if (length == 0)
    return;
  int h = 30 * channel + Offset_U + 16;
  int l = 30 * channel + Offset_U + 34;
  int b = (dataFrame->Data[offset] & mask) ? h : l;
  dataFrame->Peek(offset);
  for (int i = 0; i < length; i++)
  {
    int t = (dataFrame->Get() & mask) ? h : l;
    GUI_DrawRectangle(i * Width + Offset_L, t - 1, i * Width + Offset_L + Width, t
+ 1, YELLOW, DRAW_FULL, DOT_PIXEL_1X1);
    if (b != t)
    {
      if (b < t)
        GUI_DrawRectangle(i * Width + Offset_L, b, i * Width + Offset_L + 1, t,
YELLOW, DRAW_FULL, DOT_PIXEL_1X1);
      else
        GUI_DrawRectangle(i * Width + Offset_L, t, i * Width + Offset_L + 1, b,
YELLOW, DRAW_FULL, DOT_PIXEL_1X1);
      b = t;
    }
  }
}

//绘制按钮
void DrawButton(Button *button)
{
```

```
  GUI_DrawRectangle(button->X, button->Y, button->X + button->Width, button->Y +
button->Height, LCD_BACKGROUND, DRAW_FULL, DOT_PIXEL_1X1);
  GUI_DrawRectangle(button->X, button->Y, button->X + button->Width, button->Y +
button->Height, button->FrontColor, DRAW_EMPTY, DOT_PIXEL_2X2);
  GUI_DisString_EN(button->X + button->NameOffset_X, button->Y +
button->NameOffset_Y, button->Name, &Font16, LCD_BACKGROUND, button->FrontColor);
}

//打印地址
void PrintAddress()
{
  if(Result->End<Offset)return;
  if(Result->Start>Offset+Time)return;
  int start=max(Offset,Result->Start)-Offset;
  int end=min(Offset+Time,Result->End)-Offset;

  //进行嗅探操作，找到大小不超过绘制长度的合法显示
  sprintf(s,"Address :
0x%c%c",transcode[Result->Address>>4],transcode[Result->Address&15]);

  if((end-start)*Width>strlen(s)*12)
  {
    int o=((end-start)*Width-strlen(s)*12)/2;
    GUI_DisString_EN(Offset_L+start*Width+o, 30 * 2 + Offset_U + 18, s, &Font16,
LCD_BACKGROUND, WHITE);
  }
  else
  {
    sprintf(s,"0x%c%c",transcode[Result->Address>>4],transcode[Result->Address&15])
;
    if((end-start)*Width>strlen(s)*12)
    {
      int o=((end-start)*Width-strlen(s)*12)/2;
      GUI_DisString_EN(Offset_L+start*Width+o, 30 * 2 + Offset_U + 18, s, &Font16,
LCD_BACKGROUND, WHITE);
    }
    else
    {
      sprintf(s,"%c%c",transcode[Result->Address>>4],transcode[Result->Address&15])
;
      if((end-start)*Width>strlen(s)*12)
      {
        int o=((end-start)*Width-strlen(s)*12)/2;
        GUI_DisString_EN(Offset_L+start*Width+o, 30 * 2 + Offset_U + 18, s,
&Font16, LCD_BACKGROUND, WHITE);
```

```
      }
    }
  }

  GUI_DrawRectangle(Offset_L+start*Width, 30 * 2 + Offset_U + 16, Offset_L + end *
Width, 30 * 2 + Offset_U + 34, RED, DRAW_EMPTY, DOT_PIXEL_1X1);
}

//打印数据单元
void PrintDataRange(Data* data)
{
  if(data->End<Offset)return;
  if(data->Start>Offset+Time)return;
  int start=max(Offset,data->Start)-Offset;
  int end=min(Offset+Time,data->End)-Offset;

  sprintf(s,"Data[%d] : 0x%c%c",data-
Result->Values,transcode[data->Value>>4],transcode[data->Value&15]);

  if((end-start)*Width>strlen(s)*12)
  {
    int o=((end-start)*Width-strlen(s)*12)/2;
    GUI_DisString_EN(Offset_L+start*Width+o, 30 * 2 + Offset_U + 18, s, &Font16,
LCD_BACKGROUND, WHITE);
  }
  else
  {
    sprintf(s,"0x%c%c",transcode[data->Value>>4],transcode[data->Value&15]);
    if((end-start)*Width>strlen(s)*12)
    {
      int o=((end-start)*Width-strlen(s)*12)/2;
      GUI_DisString_EN(Offset_L+start*Width+o, 30 * 2 + Offset_U + 18, s, &Font16,
LCD_BACKGROUND, WHITE);
    }
    else
    {
      sprintf(s,"%c%c",transcode[data->Value>>4],transcode[data->Value&15]);
      if((end-start)*Width>strlen(s)*12)
      {
        int o=((end-start)*Width-strlen(s)*12)/2;
        GUI_DisString_EN(Offset_L+start*Width+o, 30 * 2 + Offset_U + 18, s,
&Font16, LCD_BACKGROUND, WHITE);
      }
    }
  }
```

```
  GUI_DrawRectangle(Offset_L+start*Width, 30 * 2 + Offset_U + 16, Offset_L + end *
Width, 30 * 2 + Offset_U + 34, WHITE, DRAW_EMPTY, DOT_PIXEL_1X1);
}

//打印所有数据
void PrintDataRanges()
{
  int r = Time * Width + 10;
  //GUI_DrawRectangle(Offset_L, 30 * 2 + Offset_U + 4, Offset_L + r + 5, 30 * 2 +
Offset_U + 16, BLACK, DRAW_FULL, DOT_PIXEL_1X1);
  if(Result!=nullptr)
  {
    PrintAddress();
    for(int i=0;i<Result->Count;i++)
      PrintDataRange(Result->Values+i);
  }
}

//绘制数据
void DrawChannels()
{
  PrintChannelLines(2);

  PrintDataFrame(0, Offset, _BV(0), Time);
  PrintDataFrame(1, Offset, _BV(1), Time);

  PrintDataRanges();
}

//刷新 GUI
void Redraw()
{
  DrawChannels();
  PrintInfo();
  DrawButtons();
  DrawCount();
}

//触摸屏调整
void Adjust()
{
  TP_Adjust();
  Redraw();
}
```

```
//I2C 采样
void I2C()
{
  Result=nullptr;
  Offset=0;
  dataFrame->Peek(0);
  SetTimer4(1599);
  Mode=1;
}

//对 I2C 进行搜索，在采样多条数据时，可依次搜索并转跳到多条数据
void Search()
{
  Offset=min(sampleInfo->Length,Offset+1);
  dataFrame->Peek(Offset);
  int last=dataFrame->Get();
  bool success=false;
  while(dataFrame->P<dataFrame->End)
  {
    int x=dataFrame->Get();
    //last = 3 SCL=1 SDA=1
    //   x = 1 SCL=1 SDA=0
    if(last==3&&x==1)
    {
      Offset=dataFrame->P-dataFrame->Data-1;
      success=true;
      break;
    }
    last=x;
  }
  if(success)
  {
    Serial.print("Search Success:");
    Serial.println(Offset);
  }
  else
  {
    Serial.println("Search Error");
    Offset=0;
  }
  Parse();
  DrawChannels();
}
```

```
//绘制长度
void DrawCount()
{
  GUI_DrawRectangle(50,300,170,320,BLACK,DRAW_FULL,DOT_PIXEL_1X1);

  sprintf(s, "Count : %d", Time);

  GUI_DisString_EN(50, 304, s, &Font16, BLACK, WHITE);
}

//增加长度
void AddCount()
{
  int i=max(0,WIndex-1);
  if(WIndex==i)return;
  WIndex=i;
  Width=Widths[WIndex];
  Time=400/Width;
  DrawCount();
  DrawChannels();
}

//减少长度
void ReduceCount()
{
  int i=min(WidthCount-1,WIndex+1);
  if(WIndex==i)return;
  WIndex=i;
  Width=Widths[WIndex];
  Time=400/Width;
  DrawCount();
  DrawChannels();
}

//设置 Column
void ChangeLine()
{
  if(Column)
  {
    Column=false;
    Button_Line->FrontColor=WHITE;
  }
  else
  {
    Column=true;
```

```
    Button_Line->FrontColor=RED;
  }
  DrawChannels();
  DrawButton(Button_Line);
}
```

Button.hpp 代码：

```
//按钮
class Button
{
public:
    //GUI
    int X;
    int Y;
    int Width;
    int Height;
    const char* Name;
    //回调
    void (*Func)();
    //文字偏移量
    int NameOffset_X;
    int NameOffset_Y;
    //前景色
    COLOR FrontColor;
    Button(int x,int y,int width,int height,const char* name,void (*func)(),int
no_x,int no_y)
    {
        X=x;
        Y=y;
        Width=width;
        Height=height;
        Name=name;
        Func=func;
        NameOffset_X=no_x;
        NameOffset_Y=no_y;
        FrontColor=WHITE;
    }
    //命中测试
    void ClickTest(int x,int y)
    {
        if(X<=x&&Y<=y&&x<=(X+Width)&&y<=(Y+Height))
        {
            Serial.println(Name);
```

```
            Func();
        }
    }
};
```

Data.hpp 代码：

```
#ifndef DATA_HPP
#define DATA_HPP

//数据帧
class DataFrame_I2C
{
public:
    //使用 Char 存储数据
    char *Data;
    //数据长度
    int Length;
    //当前指针
    char *P;
    //结束指针
    char *End;
    DataFrame_I2C(int length)
    {
        Data=new char[length];
        Length=length;
        P=Data;
        End=Data+Length;
    }
    //测试代码
    void Random()
    {
        for(int i=0;i<Length;i++)
            Set(i);
    }
    //设置当前数据，并移动
    void Set(char x)
    {
        *(P++)=x;
    }
    //获取当前数据，并移动
    char Get()
    {
        return *(P++);
```

```
    }
    //将指针移动到 offset 位置
    void Peek(int offset)
    {
        P=Data+offset;
    }
};

//采样信息
class SampleInfo
{
public:
    //数据结构大小
    size_t Size;
    //采样长度
    unsigned long Length;
    //内存大小
    unsigned long Total;
    //采样间隔 s
    double Duration;
    //采样总时间 s
    double Time;
    SampleInfo(size_t size,unsigned long  length,double duration)
    {
        Size=size;
        Length=length;
        Total=size*length;
        Duration=duration;
        Time=length*duration;
    }
};

//根据 Arduino 内存以及数据结构大小，构造采样信息
SampleInfo *GetSampleInfo(size_t size,unsigned long total)
{
    unsigned long length=total/size;
    while(length!=(length&(-length)))length-=(length&(-length));
    double duration=(OCR4A+1)/16000000.0;
    return new SampleInfo(size,length,duration);
}

//数据单元
class Data
{
public:
```

```
  //数据
  unsigned char Value;
  //应答位
  bool ACK;
  //起始位置（下降沿）
  int Start;
  //终止位置（下降沿）
  int End;
};

//数据包
class Package
{
public:
  //传输地址
  char Address;
  //读写
  bool RW;
  //应答
  bool ACK;
  //数据单元个数
  int Count;
  //数据单元
  Data *Values;
  //地址起始位置（下降沿）
  int Start;
  //地址起始位置（下降沿）
  int End;
  Package(char address,bool rw,bool ack,int count,int start,int end)
  {
    Address=address;
    RW=rw;
    ACK=ack;
    Count=count;
    Values=new Data[count];
    Start=start;
    End=end;
  }
};

#endif
```

TP_Event 代码：

```
//事件类型
typedef enum{
    None=0,
    Click=1,
    Move_U=2,
    Move_D=4,
    Move_L=8,
    Move_R=16
}TP_EVENT_Type;

//事件
class TP_EVENT
{
public:
  //起始
    uint32_t Start;
    POINT XPoint0;
    POINT YPoint0;
  //终止
    uint32_t End;
    POINT XPoint;
    POINT YPoint;
  //扫描周期数
    int ScanCount;
    TP_EVENT_Type Type;
    TP_EVENT(POINT x,POINT y)
    {
        Start=End=millis();
        XPoint0=XPoint=x;
        YPoint0=YPoint=y;
        ScanCount=1;
    }
    void Set(POINT x,POINT y)
    {
        End=millis();
        XPoint=x;
        YPoint=y;
        ScanCount++;
    }
    //更新事件类型
    void Lock()
    {
        if(ScanCount<10)Type=Click;
        else if(abs(XPoint0-XPoint)+abs(YPoint0-YPoint)<50)Type=Click;
        else
```

```
        {
            int dx=
                XPoint>XPoint0+50?1:
                    XPoint<XPoint0-50?-1:
                        0;
            int dy=
                YPoint>YPoint0+50?1:
                    YPoint<YPoint0-50?-1:
                        0;
            if(dx==1&&dy==0)Type=Move_R;
            else if(dx==-1&&dy==0)Type=Move_L;
            else if(dx==0&&dy==1)Type=Move_D;
            else if(dx==0&&dy==-1)Type=Move_U;
            else Type=None;
        }
    }
};
```

ILI9486 驱动芯片初始化代码：

```
static void LCD_InitReg(void)
{
    LCD_WriteReg(0XF9);
    LCD_WriteData(0x00);
    LCD_WriteData(0x08);

    LCD_WriteReg(0xC0);
    LCD_WriteData(0x19);//VREG1OUT POSITIVE
    LCD_WriteData(0x1a);//VREG2OUT NEGATIVE

    LCD_WriteReg(0xC1);
    LCD_WriteData(0x45);//VGH,VGL    VGH>=14V.
    LCD_WriteData(0x00);

    LCD_WriteReg(0xC2); //Normal mode, increase can change the display quality,
while increasing power consumption
    LCD_WriteData(0x33);

    LCD_WriteReg(0XC5);
    LCD_WriteData(0x00);
    LCD_WriteData(0x28);//VCM_REG[7:0]. <=0X80.

    LCD_WriteReg(0xB1);//Sets the frame frequency of full color normal mode
    LCD_WriteData(0xA0);//0XB0 =70HZ, <=0XB0.0xA0=62HZ
    LCD_WriteData(0x11);

    LCD_WriteReg(0xB4);
```

```
    LCD_WriteData(0x02); //2 DOT FRAME MODE,F<=70HZ.

    LCD_WriteReg(0xB6);//
    LCD_WriteData(0x00);
    LCD_WriteData(0x42);//0 GS SS SM ISC[3:0];
    LCD_WriteData(0x3B);

    LCD_WriteReg(0xB7);
    LCD_WriteData(0x07);

    LCD_WriteReg(0xE0);
    LCD_WriteData(0x1F);
    LCD_WriteData(0x25);
    LCD_WriteData(0x22);
    LCD_WriteData(0x0B);
    LCD_WriteData(0x06);
    LCD_WriteData(0x0A);
    LCD_WriteData(0x4E);
    LCD_WriteData(0xC6);
    LCD_WriteData(0x39);
    LCD_WriteData(0x00);
    LCD_WriteData(0x00);
    LCD_WriteData(0x00);
    LCD_WriteData(0x00);
    LCD_WriteData(0x00);
    LCD_WriteData(0x00);

    LCD_WriteReg(0XE1);
    LCD_WriteData(0x1F);
    LCD_WriteData(0x3F);
    LCD_WriteData(0x3F);
    LCD_WriteData(0x0F);
    LCD_WriteData(0x1F);
    LCD_WriteData(0x0F);
    LCD_WriteData(0x46);
    LCD_WriteData(0x49);
    LCD_WriteData(0x31);
    LCD_WriteData(0x05);
    LCD_WriteData(0x09);
    LCD_WriteData(0x03);
    LCD_WriteData(0x1C);
    LCD_WriteData(0x1A);
    LCD_WriteData(0x00);

    LCD_WriteReg(0XF1);
```

```
    LCD_WriteData(0x36);
    LCD_WriteData(0x04);
    LCD_WriteData(0x00);
    LCD_WriteData(0x3C);
    LCD_WriteData(0x0F);
    LCD_WriteData(0x0F);
    LCD_WriteData(0xA4);
    LCD_WriteData(0x02);

    LCD_WriteReg(0XF2);
    LCD_WriteData(0x18);
    LCD_WriteData(0xA3);
    LCD_WriteData(0x12);
    LCD_WriteData(0x02);
    LCD_WriteData(0x32);
    LCD_WriteData(0x12);
    LCD_WriteData(0xFF);
    LCD_WriteData(0x32);
    LCD_WriteData(0x00);

    LCD_WriteReg(0XF4);
    LCD_WriteData(0x40);
    LCD_WriteData(0x00);
    LCD_WriteData(0x08);
    LCD_WriteData(0x91);
    LCD_WriteData(0x04);

    LCD_WriteReg(0XF8);
    LCD_WriteData(0x21);
    LCD_WriteData(0x04);

    LCD_WriteReg(0X3A); //Set Interface Pixel Format
    LCD_WriteData(0x55);

}
```

计算机硬件类综合性课程设计