

**硬件课程设计报告（团队）**

**题目： 触控感应键鼠的设计**

**院（系）： 电子信息与通信学院**

**团队成员：李鑫瑞 王梦婷 吴永浩**

**指导老师： 吴建新**

**报告日期： 2024.6.12**

# 硬件课设报告

目录

[引言 4](#_Toc169187708)

[1. 项目概述 4](#_Toc169187709)

[2. 设计与实现背景 5](#_Toc169187710)

[3. 项目功能指标 5](#_Toc169187711)

[3.1 功能指标 5](#_Toc169187712)

[3.2 硬件指标 6](#_Toc169187713)

[3.3 性能指标 7](#_Toc169187714)

[3.4 软件指标 7](#_Toc169187715)

[4. 团队分工 9](#_Toc169187716)

[5. 系统框图 9](#_Toc169187717)

[5.1. 触控键盘 10](#_Toc169187718)

[5.2. 信号采集电路 10](#_Toc169187719)

[5.3. Nexys4板子 10](#_Toc169187720)

[5.4. 蓝牙发送模块 10](#_Toc169187721)

[5.5. 蓝牙接收模块 11](#_Toc169187722)

[5.6. 终端显示 11](#_Toc169187723)

[6. 各模块原理 11](#_Toc169187724)

[6.1 TTP226触控模块 11](#_Toc169187725)

[6.2 HC-08蓝牙模块 12](#_Toc169187726)

[6.3 Nexys4DDR开发板 13](#_Toc169187727)

[7. 理论分析 13](#_Toc169187728)

[8. 关键技术 14](#_Toc169187729)

[8.1 电容触控检测技术 14](#_Toc169187730)

[8.2 蓝牙通信技术 15](#_Toc169187731)

[8.3 FPGA编程技术 15](#_Toc169187732)

[8.4 蓝牙模块-端口转换技术 16](#_Toc169187733)

[8.5 上位机C#程序部分 16](#_Toc169187734)

[8.6 数据处理与优化技术 19](#_Toc169187735)

[8.7 系统集成与调试技术 19](#_Toc169187736)

[9. 模块测试 20](#_Toc169187737)

[9.1 TTP226模块测试 20](#_Toc169187738)

[9.2 HC-08模块测试 20](#_Toc169187739)

[9.3 Nexys4DDR开发板测试 21](#_Toc169187740)

[9.4 综合系统测试 22](#_Toc169187741)

[10. 仿真图 23](#_Toc169187742)

[10.1 电容式触控传感器 23](#_Toc169187743)

[10.2 HC-08蓝牙模块 24](#_Toc169187744)

[11. 器件清单 25](#_Toc169187745)

[11.1 电容式触控传感器 25](#_Toc169187746)

[11.2 FPGA NEXYS 4 ddr开发板 25](#_Toc169187747)

[11.3 蓝牙模块 25](#_Toc169187748)

[12. 总结及心得体会 25](#_Toc169187749)

[13. 参考资料 26](#_Toc169187750)

[14. 附录 26](#_Toc169187751)

[14.1 键鼠模拟程序（部分） 26](#_Toc169187752)

[14.2 键鼠模拟程序说明（部分） 32](#_Toc169187753)

[14.3 VERILOG触控板数据处理与发送代码与说明（部分） 34](#_Toc169187757)

# 引言

*21世纪初，随着智能手机和平板电脑的兴起，触控技术迎来了爆炸性的发展。2007年，苹果公司推出的iPhone采用了多点触控技术，这彻底改变了人们与电子设备交互的方式，使得触控操作更加直观和自然。多点触控技术允许用户通过多个手指同时操作屏幕，实现缩放、旋转等复杂手势。*

*随后，触控技术不断进步，出现了更多种类的触控屏，如表面声波触控屏、光学触控屏等。这些技术在教育、医疗、零售等多个领域得到广泛应用，极大地丰富了人机交互的方式。*

*如今，触控技术已经成为现代电子设备中不可或缺的一部分，赢得了广大用户的喜爱。然而，传统的物理键盘和鼠标在便携性和交互体验上仍存在一定的局限性。因此，本课程设计旨在通过实践操作，设计一款基于触控感应技术的键盘与鼠标二合一设备，旨在解决这些问题。*

## 1. 项目概述

本项目旨在利用TTP226触控模块、Nexys4DDR开发板及HC-08蓝牙发送与接收模块实现键盘-鼠标模拟功能。项目通过触控模块获取用户输入，在nexys4DDR开发板上处理触控模块输入数据，并将数据发送至蓝牙输出模块。最后数据由蓝牙接收模块收到，并通过USB数据传输线连接到pc的端口，在pc上通过代码实现对鼠标、键盘的模拟。

本项目的目标是设计并实现一个以Xilinx FPGA为核心控制器的智能控制系统，通过电容触摸板检测用户的触摸动作，并将相应的控制信号通过蓝牙模块传输到上位机或终端设备，实现以下功能：

1. **PPT翻页和文档输入**：用户可以通过触摸纸上的按键区域，远程控制PPT的翻页或在文档中输入字符。
2. **钢琴演奏**：用户可以在纸上绘制钢琴轨迹，通过触摸不同的键位，控制计算机上的钢琴程序发音，实现虚拟钢琴演奏。
3. **无线游戏手柄**：用户可以在纸上绘制游戏控制区域，通过触摸这些区域，控制游戏角色的移动和操作，实现无线游戏手柄功能。

## 2. 设计与实现背景

在现代社会中，触控技术已经广泛应用于各种电子设备中，如智能手机、平板电脑和触控笔记本等。然而，传统的物理键盘和鼠标在便携性和交互体验上存在一定的局限性。我们提出一个创新的设计——基于Xilinx FPGA以及触控和蓝牙技术的结合的触控感应键鼠，通过检测微小的电容变化，将其转换为数字信号，实现按键的触控感应。此设计不仅可以用作无线游戏，还可以一定程度上代替键盘和鼠标。

项目意义：

本项目的实施具有重要的应用价值和创新意义：

1. **低成本、高灵活性**：相较于传统的触控屏，本项目通过电容传感器和FPGA实现触控检测，成本更低，且可以根据需要自由绘制触控区域，灵活性更高。
2. **多功能一体化**：通过FPGA的强大处理能力和蓝牙模块的无线传输功能，本项目实现了多种功能的集成，用户可以在同一平台上实现PPT翻页、文档输入、钢琴演奏和游戏控制等多种操作。
3. **提升用户体验**：本项目提供了一种全新的操作方式，用户只需在纸上绘制触控区域，即可通过触摸实现对终端设备的控制，操作简单直观，提升了用户体验。

## 3. 项目功能指标

### 3.1 功能指标

1.触控感应：

检测触摸板上的按键区域，并将触摸信息转换为相应的数字信号。TTP226触控模块能够识别8个独立触摸按键，通过电容变化感应触摸，实现高灵敏度和稳定性的触控检测。

2.数据处理：

使用Nexys4DDR开发板上的FPGA对采集到的电容按键数据进行实时处理。FPGA通过编写硬件描述语言（如Verilog或VHDL）实现对数据的解析和转换，然后将处理后的数据传输到蓝牙发送模块。数据处理包括去抖动、按键状态识别和信号编码等步骤。

3.无线传输：

通过HC-08蓝牙模块，将处理后的数据无线传输到上位机（PC）。蓝牙模块使用串口通信协议，与FPGA进行数据交互，并通过蓝牙BLE标准实现稳定和低功耗的无线通信，确保数据传输的准确性和实时性。

4.游戏控制：

在上位机上，使用Unity引擎编写的程序和游戏，使得可以使用触摸板直接操控游戏。触控板的按键输入通过蓝牙传输到上位机，Unity引擎中的脚本程序接收并解析这些信号，实现游戏角色的移动、攻击等操作，增强用户的互动体验。

5.模拟键鼠：

在上位机上，使用Visual Studio平台编写C#窗体程序，处理蓝牙接收模块收到的信号。C#程序将触控板的输入信号映射为键盘和鼠标的操作，实现如下功能：

* 光标移动和点击：触控板上的特定区域可以控制鼠标光标的移动和点击操作。
* 打字输入：触控板的按键可以模拟键盘输入，实现打字功能。

这些功能使得触控板不仅可以作为输入设备，还能够在特定应用中替代传统的键盘和鼠标，提供更灵活和多样化的用户交互方式.

### 3.2 硬件指标

1.检测处理电路：

包括电容感应电路，将电容变化转换为数字信号；以及信号转换电路，将数字信号通过引脚发送至FPGA开发板。

2.FPGA控制平台：

使用Xilinx FPGA NEXYS4DDR开发板进行数据处理和控制。

3.蓝牙模块电路：

实现数据的无线传输功能。

蓝牙传输波特率：9600；

使用协议：低功耗蓝牙BLE

蓝牙发送模块连接到FPGA开发板，由其提供供电和数据输送。

蓝牙接收模块连接一块转换板为其提供供电和传输数据，并使得接收模块可以通过usb传输线连接到上位机端口。

### 3.3 性能指标

1.响应速度：

触控感应信号的响应时间应足够快，以保证流畅的用户体验。

2.精度：

能够准确检测到微小的电容变化，确保触控操作的精确性。

3.可靠性：

触控板发送的信息需要即时、无损地发送到FPGA开发板。

4.稳定性：

上位机程序需要对端口接收到的数据及时的处理、判断，确保正确的按键才能触发相应的事件；除此之外，还要考虑到用户的使用体验：应当适当的设置数据接收时间间隔，避免用户一次按键触发多次按键事件。

### 3.4 软件指标

1.上位机程序：

1.1.数据接收和处理

* **数据接收**：上位机程序使用C#编写，通过串口通信接收来自HC-08蓝牙模块的数据。程序应确保数据接收的实时性和准确性，能够处理丢包和错误数据的情况。
* **数据解析**：对接收到的数据进行解析，将其转换为相应的键盘或鼠标指令。解析过程中应考虑数据的格式和编码方式，确保正确还原触控板的操作。
* **数据处理**：处理数据包括去抖动、状态识别和动作触发等步骤，确保上位机能够准确响应触控板的输入。

1.2. 与终端设备交互：

* **模拟键盘和鼠标**：使用Windows API或其他库模拟键盘和鼠标的操作。程序应能够根据解析后的数据触发相应的键盘按键或鼠标移动、点击等操作。
* **游戏控制**：与Unity引擎编写的游戏程序进行交互，传递触控板的输入信号，实现游戏角色的控制。程序应设计良好的接口，确保数据的快速传递和响应。

2.蓝牙协议实现：

2.1.蓝牙通信协议

* **协议规范**：按照HC-08蓝牙模块的通信协议规范，实现数据的发送和接收。协议应包括连接建立、数据传输和连接断开的过程。
* **数据帧格式**：设计并实现适用于蓝牙传输的数据帧格式，包括起始位、数据位、校验位和结束位等部分，确保数据的完整性和可靠性。

2.2. 数据传输机制

* **串口通信**：在Nexys4DDR开发板上，通过UART接口与HC-08蓝牙模块进行通信。FPGA程序应能够将处理后的数据按照指定的帧格式发送至蓝牙模块。
* **数据接收与反馈**：上位机程序应能够接收并解析蓝牙模块传输的数据，并根据需要发送反馈信号，以确保数据传输的双向性和确认机制。

3. 虚拟键盘界面

3.1..界面设计

* **用户友好界面**：设计一个直观且易于操作的虚拟键盘界面。界面应包含所有标准键盘按键，并能够根据需要进行自定义布局，以适应不同的应用场景。
* **游戏控制界面**：设计一个专门用于游戏控制的界面，包含常用的游戏操作按键，如方向键、动作键等。界面应能够实时响应触控板的输入，实现流畅的游戏控制。

3.2.功能实现

* **虚拟按键映射**：将触控板的输入映射到虚拟键盘的按键上。程序应能够识别触控板的不同按键，并将其转换为相应的虚拟按键操作。
* **触控反馈**：实现触控反馈功能，在用户触摸按键时给出视觉或声音反馈，增强用户体验。例如，当用户点击虚拟按键时，按键会在界面上显示按下状态，同时播放点击音效。
* **可配置性**：提供界面的自定义配置功能，使用户能够根据个人需求调整按键布局和功能。例如，用户可以自定义某个按键的功能或位置，以适应特定的应用或游戏需求。

## 4. 团队分工

* **上位机编程----吴永浩**：

负责整体系统设计和架构规划。

设计和实现蓝牙模块的集成。

完成电脑端键鼠输入信号处理的设计。

* **FPGA编程----李鑫瑞**：

负责触控板信号的输入处理。

负责触控板性能的升级。

搭建测试平台和环境，对系统进行验证和调试。

* **系统调试----王梦婷**：

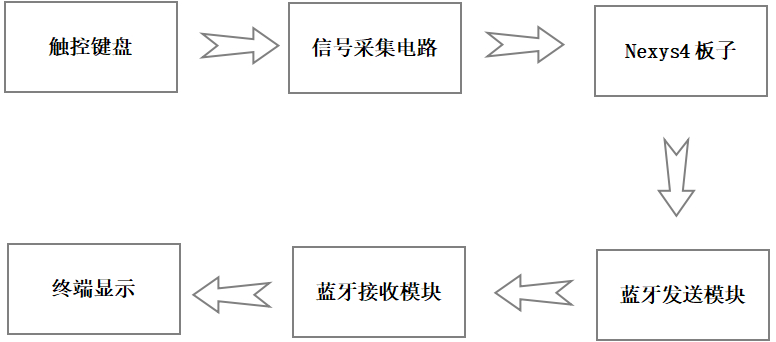
负责处理触控板输入信号并传输至蓝牙发送端。

负责对硬件和软件进行系统联调。

搭建测试平台和环境，对系统进行验证和调试。

## 5. 系统框图

在本项目中，我们设计了一个高度集成的系统框图，明确了各个模块之间的数据流向和控制逻辑。系统框图不仅展示了系统的整体结构，还体现了模块化设计的优越性，便于后续的维护和升级。



其中信号采集电路已集成到触控键盘模块。

**系统框图详细描述**

### 5.1. 触控键盘

触控键盘（TTP226模块）是整个系统的输入设备。用户通过触摸键盘上的按键来输入指令。触控键盘通过电容变化感应触摸信号，并将这些信号转换为相应的电信号输出。

### 5.2. 信号采集电路

信号采集电路负责接收来自触控键盘的电信号，并对这些信号进行初步处理。该电路可以包括信号放大、滤波和模数转换等功能，以确保信号的准确性和稳定性。处理后的信号将传输到Nexys4板子进行进一步处理。

### 5.3. Nexys4板子

Nexys4板子是整个系统的核心处理单元。该板子基于Xilinx Artix-7 FPGA，具有强大的处理能力和丰富的外设接口。Nexys4板子接收并处理来自信号采集电路的触控信号，通过FPGA编写的程序实现对信号的解析、处理和编码。处理后的数据将通过串口发送到蓝牙发送模块。

### 5.4. 蓝牙发送模块

蓝牙发送模块（HC-08模块）与Nexys4板子通过串口通信。它负责将处理后的数据无线传输到蓝牙接收模块。该模块遵循蓝牙4.0协议，具有低功耗和长传输距离的特点，确保数据的稳定传输。

### 5.5. 蓝牙接收模块

蓝牙接收模块接收来自蓝牙发送模块的数据。它将无线接收的数据转换为串口信号，传输到终端显示设备进行进一步处理和显示。蓝牙接收模块同样遵循蓝牙4.0协议，确保数据接收的准确性和实时性。

### 5.6. 终端显示

终端显示设备（上位机）接收并处理来自蓝牙接收模块的数据。上位机运行一个C#编写的程序，该程序负责解析接收到的数据，并根据这些数据执行相应的操作，如控制鼠标光标移动、模拟键盘按键等。此外，终端显示设备还可以运行Unity引擎编写的游戏，通过触控键盘控制游戏中的角色和操作。

## 6. 各模块原理

### 6.1 TTP226触控模块

TTP226是一款电容式触摸检测集成电路（IC），它能够提供多达8个触摸键，并且设计用来取代传统的固定尺寸触摸板或按钮。这款IC具备低功耗和宽工作电压的特点，使其适合在直流（DC）或交流（AC）应用中使用。TTP226的工作原理基于电容式感应，能够检测到人手的接近或触摸，从而触发相应的按键动作。它广泛应用于电子设备中，如遥控器、家用电器、工业控制设备等，以提供用户友好的交互界面。當VDD=3V時System clock約為440KHz,大約55Hz更新輸出一次。大多数输出模式工作在直接(direct)或串行(serial)模式。只有当OPS0=0时，输出模式会为矩阵(matrix)类型。

其还集成了信号处理模块，从触摸输入中读取数据，经过处理后通过UART发送出去。

顶层模块 top\_module：

**A.时钟分频器和状态机**

计数器 clock\_counter: 用于生成一定间隔的时间信号，这里定义为23位宽，可以计数到5000000。

参数 CLOCK\_DIVIDE: 设置为5000000，用于100MHz时钟频率下生成约100ms的时间间隔。

**B.状态机: 在时钟上升沿或重置信号有效时进行状态转换。**

当重置信号有效时，将计数器置零，tx\_data和tx\_data\_valid也置零。否则，计数器自增，当计数到CLOCK\_DIVIDE时：将touch\_output扩展为8位并赋值给tx\_data（高4位置零）。tx\_data\_valid置为1，表示数据有效。

若未达到CLOCK\_DIVIDE，则计数器继续自增，tx\_data\_valid保持为0。

#### 工作步骤：

1. **电容感应**：每个按键区域有一个电容，当用户触摸时，电容值发生变化。
2. **信号处理**：TTP226内部的电路检测电容变化，并将其转换为电信号。
3. **输出信号**：处理后的信号通过并行输出引脚传输，确保各个按键信号的准确性。

### 6.2 HC-08蓝牙模块

HC-08是一款低功耗蓝牙模块，支持蓝牙4.0 BLE协议。系统中使用了两个HC-08模块，分别作为数据的发送和接收端。发送端HC-08模块通过串口与Nexys4DDR开发板连接，接收端HC-08模块通过串口-USB转换架连接到PC端口的USB传输线。数据传输过程中，TXD引脚负责发送数据，RXD引脚负责接收数据。HC-08模块在BLE协议下进行通信，具有低功耗、传输距离长和数据传输稳定的特点。

HC-08模块因其易用性、成本效益和广泛的应用而受到电子爱好者和专业开发者的欢迎。

其部分主要引脚及其功能如下表所示：

|  |  |
| --- | --- |
| STATE | 连接状态引脚(未连接低电平,连接后输出高电平) |
| RXD | 串口输入引脚（TTL 电平） |
| TXD | 串口输出引脚（TTL 电平） |
| GND | 地 |
| VCC | 电源(1.8-3.6V) |

#### 工作步骤：

1. **初始化**：HC-08模块上电后初始化，设置波特率、角色（主/从）等参数。
2. **连接建立**：发送端与接收端通过蓝牙协议进行配对和连接建立。
3. **数据传输**：发送端通过TXD引脚发送数据，接收端通过RXD引脚接收数据。数据以串口数据帧格式进行传输，包括起始位、数据位、校验位和停止位。
4. **连接断开**：数据传输完成后，蓝牙模块可以按照协议断开连接，进入低功耗模式。

#### 硬件连接：

* **发送端**：HC-08发送模块的TXD引脚连接到Nexys4DDR开发板的UART接收引脚（RX），RXD引脚连接到Nexys4DDR开发板的UART发送引脚（TX）。
* **接收端**：HC-08接收模块的TXD引脚连接到串口-USB转换架的RX引脚，RXD引脚连接到串口-USB转换架的TX引脚。串口-USB转换架通过USB传输线连接到PC端口。

### 6.3 Nexys4DDR开发板

Nexys4DDR开发板是一款基于Xilinx Artix-7 FPGA的开发板，提供丰富的外设接口和强大的逻辑资源。这款开发板的核心是一块Xilinx Artix-7 XC7A200T FPGA芯片，它拥有高达200万逻辑单元和716百万门的逻辑能力，为用户提供了强大的数据处理能力。此外，512MB的DDR3 SDRAM内存保证了高速数据处理和存储的需求。该开发板具有以下主要功能和特点：

* **FPGA资源**：内置Xilinx Artix-7 FPGA，具有强大的逻辑处理能力，支持用户编写Verilog或VHDL代码实现复杂的硬件逻辑功能。
* **外设接口**：板载提供多种外设接口，包括UART、SPI、I2C、GPIO等，可以与各种外部设备进行连接和通信。
* **存储资源**：板载具有DDR2 SDRAM、Flash存储器等，支持数据的存储和访问。
* **开发环境**：支持Xilinx的Vivado设计套件，通过编写硬件描述语言，实现FPGA内部逻辑功能。

在本项目中，Nexys4DDR开发板负责接收和处理来自TTP226触控模块的触摸信号。通过编写FPGA代码，实现对触摸信号的解析、处理和编码。处理后的数据通过UART接口传输到HC-08蓝牙发送模块，实现数据的无线传输。Nexys4DDR开发板在整个系统中充当核心处理单元，负责信号的实时处理和传输。

## 7. 理论分析

项目的核心在于电容触控检测和蓝牙通信的实现。电容触控检测依赖于TTP226模块，通过检测电容变化来感应触摸信号。电容触控技术的基本原理是，当用户的手指靠近或接触电容触摸传感器时，会改变传感器表面的电场分布，导致电容值的变化。TTP226模块内部集成了高灵敏度的电容检测电路，能够实时感应这些变化，并将其转换为电信号输出。触控信号通过并行引脚传输，确保各个按键的触摸信号能够被准确区分和传输。

信号经TTP226处理后，通过信号采集电路传输至Nexys4DDR开发板。Nexys4DDR基于Xilinx Artix-7 FPGA，提供强大的逻辑处理能力和丰富的外设接口。FPGA通过编写Verilog或VHDL代码，实现对触控信号的进一步处理。处理过程包括信号去抖动、按键状态识别和信号编码等步骤。去抖动技术用于消除按键抖动造成的误触发，确保信号的准确性；按键状态识别则用于判断按键的按下和松开状态；信号编码则将处理后的触控信号转换为特定的格式，便于后续的传输和解析。

处理后的信号通过UART接口传输至HC-08蓝牙发送模块。HC-08是一款低功耗蓝牙模块，支持蓝牙4.0 BLE协议。蓝牙通信的实现需要遵循特定的协议规范，确保数据的稳定传输。HC-08模块通过串口与Nexys4DDR开发板连接，利用TXD和RXD引脚进行数据的发送和接收。数据传输过程中，TXD引脚负责将处理后的触控信号发送至蓝牙接收模块，而RXD引脚则用于接收来自蓝牙接收模块的反馈信号。

蓝牙接收模块通过BLE协议接收到的数据通过串口-USB转换架传输至PC端口。BLE（Bluetooth Low Energy）协议是蓝牙4.0标准中的一个重要部分，具有低功耗、传输距离长和数据传输稳定的特点，适用于低功耗设备间的无线通信。串口-USB转换架将蓝牙接收模块的串口信号转换为USB信号，方便PC端口进行数据接收和处理。

在PC端，使用C#编写的上位机程序负责接收和处理来自蓝牙接收模块的数据。程序通过串口通信接收数据，并对数据进行解析，将其转换为相应的键盘或鼠标指令。解析后的数据可以用于控制鼠标光标的移动、点击和键盘的按键操作。此外，PC端还可以运行使用Unity引擎编写的游戏程序，通过触控信号实现对游戏角色的控制，增强用户的互动体验。

项目的实现涉及电容触控检测、信号处理、蓝牙通信和数据解析等多个环节。各个模块的协同工作，确保了系统的稳定性和实时性，实现了触控键盘的功能扩展和应用。通过合理的设计和实现，本项目在用户交互和应用扩展方面具有重要意义。

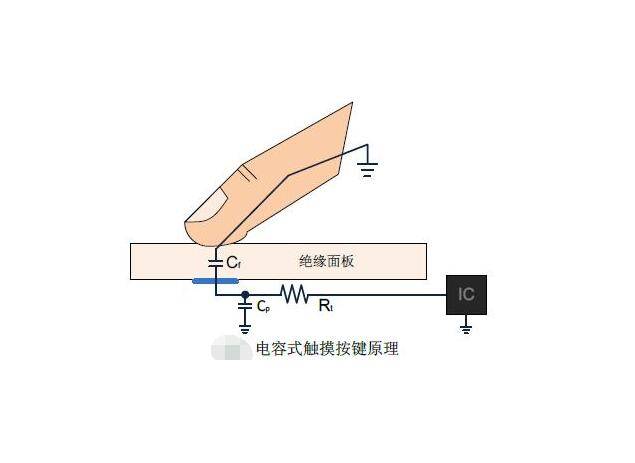
## 8. 关键技术

### 8.1 电容触控检测技术

电容触控原理基于电容变化的检测。电容是一种存储电荷的能力，而电容触控屏由多层材料构成，其中包括一个导电的触控层，通常覆盖在显示屏的表面。

当用户的手指接触到触控屏时，由于人体是导电的，手指会改变触控屏表面的电容分布。这种变化会导致电荷的重新分布，从而在手指接触点附近形成一个电容值的变化。

TTP226模块利用电容变化检测触摸信号，具有高灵敏度和稳定性。电容触控技术的核心在于电容值的变化。当用户触摸触控面板时，电容值会发生变化，TTP226模块通过检测这些变化来识别触摸事件。该模块内部集成了高精度的电容检测电路，能够实时感应微小的电容变化，并将其转换为数字信号输出。高灵敏度的电容检测技术确保了触控操作的准确性和响应速度，适用于各种触控应用场景。此外，TTP226模块具有较高的抗干扰能力，能够在复杂电磁环境下稳定工作，保证触控信号的可靠性。通过合理的电路设计和优化的信号处理算法，TTP226模块在实际应用中展现出了卓越的性能。



### 8.2 蓝牙通信技术

HC-08模块采用蓝牙BLE（Bluetooth Low Energy）标准，具有低功耗和较长的通信距离。BLE技术特别适合低功耗设备间的短距离通信，通过简化的协议栈和优化的传输机制，显著降低了功耗。HC-08模块支持多种工作模式，包括广播模式和连接模式，能够灵活适应不同的应用需求。

在本项目中，两个HC-08模块分别作为发送端和接收端，通过串口与其他设备连接，实现了稳定可靠的数据传输。蓝牙通信的实现需要遵循严格的协议规范，确保数据传输的完整性和准确性。通过合理的协议栈设计和优化的传输机制，HC-08模块在低功耗和高效传输之间找到了平衡点，适用于各种无线通信场景。

### 8.3 FPGA编程技术

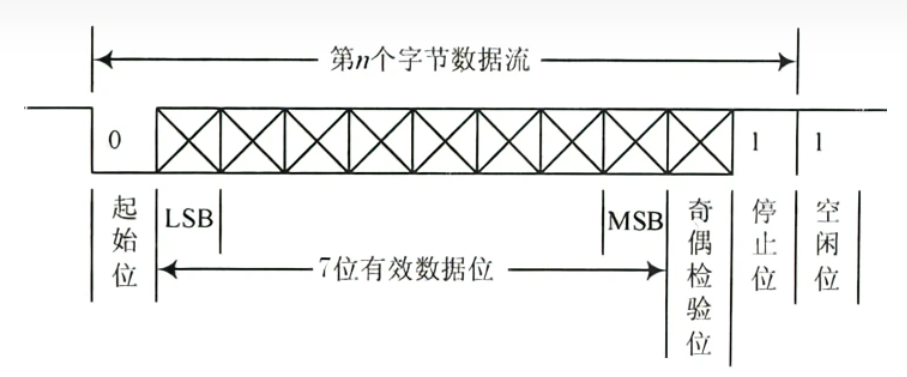
Nexys4DDR开发板基于Xilinx Artix-7 FPGA，具有强大的逻辑处理能力和丰富的外设接口。通过编写Verilog或VHDL硬件描述语言，可以实现复杂的硬件逻辑功能。

在本项目中，FPGA负责接收并处理来自TTP226触控模块的并行信号，将其转换为串行信号，并以一定的时间间隔发送到蓝牙发送模块的输入端口。FPGA编程技术在信号处理、数据传输和逻辑控制方面发挥了关键作用，确保了系统的稳定性和实时性。通过灵活的硬件编程，FPGA能够实现高度并行的信号处理和实时的数据传输，为系统的性能优化提供了强有力的支持。FPGA的高可编程性使其在各种复杂应用场景中得到了广泛应用，展现出了优异的适应性和扩展性。

### 8.4 蓝牙模块-端口转换技术

蓝牙接收模块通过串口-USB转换架连接到PC的USB端口，实现了蓝牙到USB端口的转换。转换模块将蓝牙接收模块接收到的串口信号转换为USB信号，使PC能够通过COM端口接收数据。该转换技术不仅扩展了蓝牙模块的应用范围，还简化了与PC的连接方式，确保了数据传输的便捷性和兼容性。串口-USB转换架在数据传输过程中起到了桥梁作用，通过有效的数据帧格式转换和传输协议匹配，确保了数据传输的稳定性和高效性。该技术在实际应用中具有广泛的适用性，能够满足各种数据传输需求，提升了系统的兼容性和灵活性。

数据通信格式如下图所示，一个UART数据包由10个数据位组成，包含1位起始位，7位有效数据位，1位奇偶校验位和1位停止位。UART串口信号线上空闲时常驻高电平，当检测到低电平下降沿时认为数据传输开始，到停止位时数据传输结束，一共10位数据位组成一个数据包。



下表为通信数据包的主要内容位类型：

|  |  |
| --- | --- |
| 起始位 | 通信线路上空闲时为“1”，当检测到“0”即下降沿时，认为数据传输开始。 |
| 有效数据位 | 传输开始后传递的需要接收和发送的数据值，可以表示指令或数据。 |
| 奇偶校验位 | 奇偶校验，通过校验传输数据中“1”的个数为奇数个（奇校验）或偶数个（偶校验）来指示传输数据是否正确。 |
| 停止位 | 数据传输结束，传输线恢复常“1”状态。 |

### 8.5 上位机C#程序部分

在C#中，可以使用`SendKeys`类和`mouse\_event`函数来实现键盘和鼠标的模拟操作。以下是这两种方法的简介：

1.SendKeys 类

SendKeys类是.NET Framework的一部分，用于发送按键到活动窗口。它模拟了键盘输入，可以用于发送单个字符、字符串或特殊键（如箭头键、功能键等）。

\*\*使用 SendKeys 的示例代码\*\*：

public class Program

{

[STAThread]

public static void Main()

{ // 模拟按下字母 'a' 键

SendKeys.Send("a");

// 模拟按下 Enter 键

SendKeys.Send("{ENTER}");

// 模拟按下箭头键

SendKeys.Send("{LEFT}"); // 向左箭头

SendKeys.Send("{RIGHT}"); // 向右箭头

}

}

SendKeys主要用于发送按键到活动窗口，它不会模拟按键的按下和释放动作，而是将按键作为输入发送到当前活动的应用程序。

2.mouse\_event 函数

`mouse\_event`函数是一个Windows API函数，用于模拟鼠标的各种动作，包括移动、点击、滚轮滚动等。使用P/Invoke可以将其引入C#程序中。

\*\*使用 mouse\_event 的示例代码\*\*：

```csharp

using System;

using System.Runtime.InteropServices;

public class Program

{

// Windows API 声明

[DllImport("user32.dll")]

private static extern void mouse\_event(uint dwFlags, int dx, int dy, uint dwData, UIntPtr dwExtraInfo);

private const uint MOUSEEVENTF\_LEFTDOWN = 0x02; // 鼠标左键按下

private const uint MOUSEEVENTF\_LEFTUP = 0x04; // 鼠标左键抬起

private const uint MOUSEEVENTF\_MOVE = 0x0001; // 鼠标移动

[STAThread]

public static void Main()

{

// 模拟鼠标移动到屏幕坐标 (100, 100)

mouse\_event(MOUSEEVENTF\_MOVE, 100, 100, 0, UIntPtr.Zero);

// 模拟鼠标左键按下

mouse\_event(MOUSEEVENTF\_LEFTDOWN, 100, 100, 0, UIntPtr.Zero);

// 模拟鼠标左键抬起

mouse\_event(MOUSEEVENTF\_LEFTUP, 100, 100, 0, UIntPtr.Zero);

}

}

mouse\_event函数的dwFlags参数定义了要模拟的鼠标事件类型，`dx`和`dy`参数定义了鼠标移动的相对或绝对坐标，`dwData`通常用于表示鼠标滚轮的滚动量，`dwExtraInfo`是一个额外信息的指针，通常设置为`UIntPtr.Zero`。dx：指定鼠标沿x轴的绝对位置或者从上次鼠标事件产生以来移动的数量，依赖于MOUSEEVENTF\_ABSOLUTE的设置。给出的绝对数据作为鼠标的实际X坐标；给出的相对数据作为移动的mickeys数。一个mickey表示鼠标移动的数量，表明鼠标已经移动。dy：指定鼠标沿y轴的绝对位置或者从上次鼠标事件产生以来移动的数量，依赖于MOUSEEVENTF\_ABSOLUTE的设置。给出的绝对数据作为鼠标的实际y坐标，给出的相对数据作为移动的mickeys数。

### 8.6 数据处理与优化技术

在整个系统中，数据处理与优化技术贯穿始终。TTP226触控模块产生的触摸信号需要经过FPGA的处理，包括去抖动、按键状态识别和信号编码等步骤。通过去抖动技术，可以有效消除按键抖动带来的误触发问题，保证信号的稳定性和准确性。按键状态识别技术则用于判断按键的按下和松开状态，确保每一次触控操作都能够被准确识别。信号编码技术将处理后的触摸信号转换为特定的格式，便于后续的传输和解析。在蓝牙通信过程中，数据传输需要经过协议栈的处理，确保传输的完整性和准确性。通过优化传输机制和数据帧格式，可以有效提升数据传输的效率和稳定性。

### 8.7 系统集成与调试技术

项目的成功实现离不开各个模块的系统集成与调试。在系统集成过程中，需要将TTP226触控模块、Nexys4DDR开发板、HC-08蓝牙模块和上位机程序进行有效的连接和协调。通过合理的系统架构设计和接口定义，确保各个模块之间的数据传输和信号处理能够顺畅进行。在系统调试过程中，通过逐步验证和优化各个模块的功能，发现并解决潜在的问题，确保系统的整体性能和稳定性。系统集成与调试技术在项目开发中起到了关键作用，确保了各个模块的协同工作和系统的最终实现。

## 9. 模块测试

### 9.1 TTP226模块测试

#### 测试方法

TTP226触控模块的测试主要通过单片机读取触控按键信号，并将这些信号通过串口输出到PC上进行验证。具体步骤如下：

1. 将TTP226模块连接到单片机的输入引脚，确保每个触摸按键的输出引脚都连接到单片机的对应引脚。
2. 编写单片机程序，实时读取每个触摸按键的状态，并将这些状态数据通过串口发送到PC。
3. 在PC上编写一个简单的串口接收程序，接收并显示来自单片机的按键状态数据。
4. 手动按下TTP226模块上的每个触摸按键，观察PC端显示的按键状态变化，验证触摸按键的响应情况。

#### 测试结果

经过多次测试，所有触摸按键都能正常响应，且输出数据准确。具体测试结果如下：

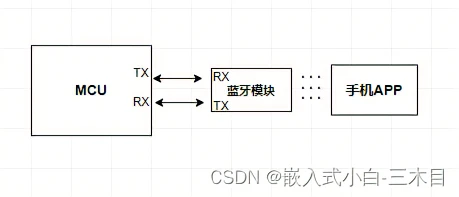
* 每个触摸按键在被按下时，PC端显示的按键状态会立即发生变化，表明触控信号能够实时传输。
* 按键状态的变化与实际按键操作完全一致，没有出现误触发或漏触发的情况。
* 在不同的触摸力度和环境条件下，TTP226模块的灵敏度和稳定性表现良好，能够准确检测到每一次触摸操作。

### 9.2 HC-08模块测试

#### 测试方法

HC-08蓝牙模块的测试主要通过与手机蓝牙配对，发送和接收数据进行验证。具体步骤如下：

1. 将HC-08蓝牙模块与单片机连接，通过串口进行通信。
2. 编写单片机程序，设置HC-08模块进入配对模式，并等待手机蓝牙的连接请求。
3. 使用手机蓝牙功能搜索HC-08模块，并进行配对连接。
4. 在手机上编写一个简单的蓝牙通信应用，发送数据到HC-08模块，并接收从HC-08模块发送的数据。
5. 通过单片机程序发送测试数据到手机，并接收手机发送的数据，验证数据传输的正确性和稳定性。



*测试通信示意图*

#### 测试结果

经过多次测试，HC-08蓝牙模块的通信稳定，数据传输准确。具体测试结果如下：

* 手机能够成功搜索到HC-08蓝牙模块，并顺利完成配对连接。
* 手机应用发送的数据能够被HC-08模块接收，并通过串口传输到单片机，单片机能够正确处理接收到的数据。
* 单片机发送的数据能够通过HC-08模块传输到手机，手机应用能够准确接收并显示这些数据。
* 在不同的通信距离和环境条件下，蓝牙通信的稳定性和传输速度表现良好，没有出现数据丢失或传输错误的情况。

### 9.3 Nexys4DDR开发板测试

#### 测试方法

Nexys4DDR开发板的测试主要通过编写简单程序，测试开发板的输入输出功能进行验证。具体步骤如下：

1. 编写一个简单的Verilog或VHDL程序，控制开发板上的LED灯和按键。
2. 将程序下载到Nexys4DDR开发板上，观察开发板的运行情况。
3. 手动操作开发板上的按键，观察LED灯的状态变化，验证按键输入和LED输出的正确性。
4. 编写更复杂的程序，测试开发板的其他外设接口，如VGA、UART、SPI等，验证其功能和性能。

#### 测试结果

经过多次测试，Nexys4DDR开发板的各项功能正常，能够正确执行程序。具体测试结果如下：

* 简单程序测试中，开发板上的LED灯能够根据按键的操作变化状态，按键输入和LED输出的响应速度和准确性都表现良好。
* 更复杂的程序测试中，开发板的其他外设接口，如VGA、UART、SPI等，均能够正常工作，数据传输和处理稳定。
* 在不同的测试场景和条件下，Nexys4DDR开发板的稳定性和性能表现出色，能够胜任各种复杂应用需求。

### 9.4 综合系统测试

#### 测试方法

综合系统测试旨在验证各模块集成后的整体性能。具体步骤如下：

1. 将TTP226触控模块、Nexys4DDR开发板、HC-08蓝牙模块以及上位机程序进行系统集成。
2. 编写FPGA程序，处理来自TTP226模块的触控信号，并通过HC-08模块将数据传输到PC。
3. 在PC上运行C#上位机程序，接收并处理蓝牙模块传输的数据，模拟键鼠功能。
4. 手动操作触控模块，观察PC上的鼠标和键盘响应情况，验证整体系统的性能和稳定性。

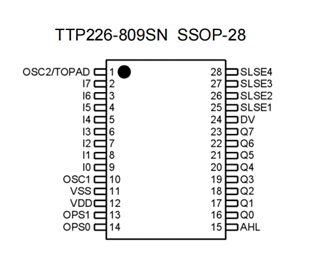
#### 测试结果

综合系统测试结果表明，各模块集成后的整体系统性能良好。具体测试结果如下：

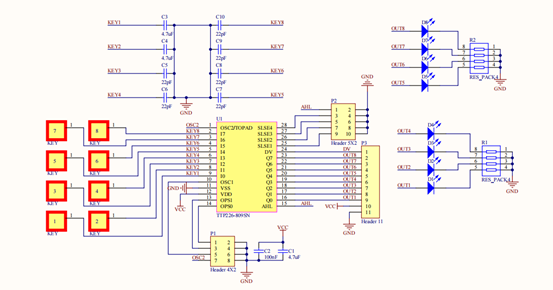
* 触控模块的触摸信号能够准确传输到FPGA，并通过蓝牙模块发送到PC。
* PC端的上位机程序能够实时接收并处理来自蓝牙模块的数据，模拟键鼠功能。
* 系统在不同的操作条件下表现稳定，没有出现数据传输延迟或错误的情况。
* 整体系统的响应速度快，用户操作体验良好，能够满足实际应用需求。

## 10. 仿真图

### 10.1 电容式触控传感器

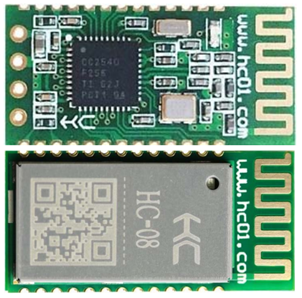


*TTP226模块芯片封装*

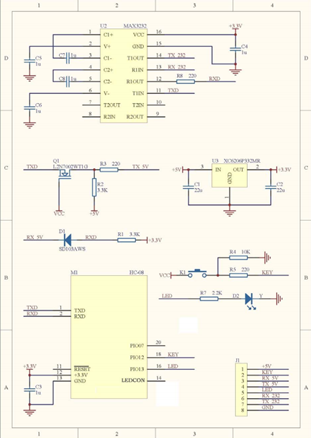


*TTP226电路图*

### 10.2 HC-08蓝牙模块



*HC-08蓝牙模块实物图*



*HC-08蓝牙模块电路图*

## 11. 器件清单

### 11.1 电容式触控传感器

一块TTP226 8路电容式触摸开关模块，及其连接电路

### 11.2 FPGA NEXYS 4 ddr开发板

使用Xilinx nexys4DDR开发板，进行电源和接口设计。

编写FPGA逻辑程序，完成电容模块数据的处理。

设计与蓝牙模块的数据交互，确保与蓝牙模块的通信稳定。

### 11.3 蓝牙模块

选择两块HC-08主从一体蓝牙模块分别作为发送模块和接收模块。

设计接口电路，实现与FPGA以及上位机（个人电脑）的无缝连接。

## 12. 总结及心得体会

通过本次项目，我们深入了解了电容触控技术和蓝牙通信技术的原理及应用。电容触控技术通过检测电容变化来识别触摸操作，这种技术在触控设备中具有广泛的应用前景。我们通过TTP226模块，成功实现了高灵敏度的触摸检测，并将触摸信号转换为数字信号进行处理。在蓝牙通信技术方面，HC-08模块采用低功耗蓝牙（BLE）标准，通过串口与其他设备进行通信，实现了数据的稳定传输。我们通过该模块，成功实现了蓝牙数据的发送和接收，将触控信号传输到上位机。

项目中，我们面对了硬件连接、驱动编写、通信协议实现等多方面的挑战。在硬件连接方面，我们需要确保各个模块之间的信号传输稳定，特别是TTP226模块与FPGA开发板、HC-08模块与单片机之间的连接。在驱动编写方面，我们需要编写FPGA程序来处理触控信号，并通过串口发送数据。在通信协议实现方面，我们需要确保蓝牙模块能够稳定进行数据传输，并在上位机中正确接收和处理数据。

在项目的不同阶段，我们遇到了许多问题。例如，在硬件连接过程中，由于电路设计的不完善，曾出现触控信号不稳定的情况。通过不断尝试和调整电路设计，我们最终解决了这一问题。在驱动编写过程中，由于对FPGA编程不够熟练，曾遇到程序运行不正常的情况。通过查阅资料，我们逐渐掌握了FPGA编程的技巧，成功编写了稳定的驱动程序。在通信协议实现过程中，由于对蓝牙通信协议不够了解，曾出现数据传输不稳定的情况。通过深入研究BLE协议，并进行多次调试，我们最终实现了稳定的蓝牙数据传输。

团队合作在项目中发挥了关键作用。每个成员都充分发挥了自己的专长，共同解决了项目中遇到的各种问题。在项目的初期阶段，我们通过分工协作，明确了各自的任务，并定期进行交流和讨论，确保项目进展顺利。在项目的实施过程中，我们通过协作解决了硬件连接、驱动编写和通信协议实现等方面的问题。每次遇到困难时，团队成员都积极参与讨论，共同寻找解决方案。通过团队合作，我们不仅提升了项目的效率，也增进了彼此的理解和信任。

此次项目经历不仅提升了我们的硬件设计与调试能力，也增强了我们对团队合作重要性的认识。在硬件设计方面，我们通过实践掌握了电容触控技术和蓝牙通信技术的原理及应用，提高了电路设计和信号处理的能力。在调试方面，我们通过多次调试和优化，解决了项目中遇到的各种问题，增强了故障排除和系统优化的能力。在团队合作方面，我们通过分工协作和共同努力，顺利完成了项目，提高了团队协作和沟通的能力。

这次项目不仅是一次技术上的提升，更是一次宝贵的学习和成长经历。我们不仅学到了很多新的知识和技能，还在项目中体验到了团队合作的力量。通过这个项目，我们更加深刻地认识到，在实际工程项目中，技术和团队合作同样重要。只有将技术和团队合作相结合，才能成功完成复杂的工程项目。我们也认识到，不断尝试和学习是解决问题的关键，只有通过不断的努力和探索，才能在技术领域不断进步。此次项目经历为我们的学习和职业发展打下了坚实的基础，也为我们今后的学习和工作提供了宝贵的经验和启示。

## 13. 参考资料

* + TTP226-809SN 8按键触摸检测芯片的详细中文数据手册
  + Nexys 4 DDR Reference Manua
  + Nexys4 开发板用户手册
  + Unity User Manual 2022.3 (LTS)
  + HC-08蓝牙传输模块用户手册
  + MicroSoft Visual Studio 文档

## 14. 附录

### 14.1 键鼠模拟程序（部分）

此部分为键鼠模拟程序的部分代码及其解释，省略了按钮等控件部分。

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO.Ports;

using System.Threading;

namespace comController

{

    public partial class Form1 : Form

    {

        private string portName = "COM5";

        private int baudRate = 9600;

        private Parity parity = Parity.Odd;//奇校验

        private int dataBits = 8;

        private StopBits stopBits = StopBits.One;//停止位

        private void clear\_Click(object sender, EventArgs e)

        {

            textBox1.Clear();

        }

        private void DisConnect\_Click(object sender, EventArgs e)

        {

            ClosePort();

        }

        private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

        {

            canRecieveMsg = false;

            ClosePort();

        }

 // 串口对象

        private SerialPort sp = null;//创建一个串口对象

        // 读取数据线程对象

        private Thread dataReceiveThread = null;//创建一个线程

        private Thread writeReceived = null;//创建一个线程,用于写入接收到的数据

        // 是否接收数据

        private bool canRecieveMsg = true;

        // 接收到的数据

        public string strReceived;//接收到的数据

        public byte[] bytesReceived;//接收到的字节数组

        public int bytesLength;//接收到的字节数组长度

        private bool IsOpenSerial = false;

        public string receivedData = string.Empty;

        [System.Runtime.InteropServices.DllImport("user32")]//引入user32.dll鼠标库

        private static extern int mouse\_event(int dwFlags, int dx, int dy, int cButtons, int dwExtraInfo);

        //移动鼠标

        const int MOUSEEVENTF\_MOVE = 0x0001;

        //模拟鼠标左键按下

        const int MOUSEEVENTF\_LEFTDOWN = 0x0002;

        //模拟鼠标左键抬起

        const int MOUSEEVENTF\_LEFTUP = 0x0004;

        //模拟鼠标右键按下

        const int MOUSEEVENTF\_RIGHTDOWN = 0x0008;

        //模拟鼠标右键抬起

        const int MOUSEEVENTF\_RIGHTUP = 0x0010;

        //模拟鼠标中键按下

        const int MOUSEEVENTF\_MIDDLEDOWN = 0x0020;

        //模拟鼠标中键抬起

        const int MOUSEEVENTF\_MIDDLEUP = 0x0040;

        //标示是否采用绝对坐标

        const int MOUSEEVENTF\_ABSOLUTE = 0x8000;

        public int moveSpeed = 10;//鼠标移动速度

        public int moveSleep = 30;//鼠标移动延时

        public int keybdSleep = 15;//键盘输入延时

        public int controlMode = 0;//控制模式，0为鼠标，1为键盘

        public Form1()

        {

            InitializeComponent();

            Control.CheckForIllegalCrossThreadCalls = false;

        }

        private void Form1\_Load(object sender, EventArgs e)//窗体加载时

        {

            stateTEXT.Text = "COM5";

            textBox1.Text = "9600";

            sp = new SerialPort(portName, baudRate, parity, dataBits, stopBits);

            strReceived = string.Empty;

        }

        public void ClosePort()

        {

            try

            {

                sp.Close();

                Console.WriteLine("close success");

                stateTEXT.Text = "关闭成功";

                //Debug.Log("close success");

                dataReceiveThread.Abort();

                writeReceived.Abort();

            }

            catch (System.Exception ex)

            {

                Console.WriteLine(ex.Message);

                //Debug.Log(ex.Message);

            }

        }

…………

        void DataReceiveFunction()

        {

            try

            {

                while (canRecieveMsg)

                {

                    // 设定读取间隔

                    Thread.Sleep(25);//25ms读取一次

                    if (!sp.IsOpen)

                        return;

                    int datalength = sp.BytesToRead;

                    if (datalength == 0)

                    {

                        bytesLength = 0;

                        continue;

                    }

                    byte[] bytes = new byte[datalength];

                    bytesReceived = new byte[datalength];

                    bytesLength = datalength;

                    sp.Read(bytes, 0, datalength);

                    bytes.CopyTo(bytesReceived, 0);

                    //strReceived = System.Text.Encoding.Default.GetString(bytes);//将字节数组转换为字符串

                    textBox1.Text = BitConverter.ToString(bytesReceived);

                    if(controlMode == 0)

                    {

                        nowMODE.Text = "鼠标模式";

                        mousecontroller(bytesReceived);

                    }

                    else if(controlMode == 1)

                    {

                        nowMODE.Text = "键盘模式";

                         keyboardcontroller(bytesReceived);

                    }

                    //receivedData = bytesReceived.ToString();

                }

            }

            catch (System.Exception ex)

            {

                if (ex.GetType() != typeof(ThreadAbortException))

                {

                }

                Console.WriteLine(ex.Message);

                //Debug.Log(ex);

            }

        }

……

        private void mousecontroller(byte[] comIN)//控制鼠标

            if (comIN[0] == 0x05)//左键单击，按下数字键4

            {

                mouse\_event(MOUSEEVENTF\_LEFTDOWN | MOUSEEVENTF\_LEFTUP, 0, 0, 0, 0);

            }

            else if (comIN[0] == 0x02)//右键单击，2

            {

                mouse\_event(MOUSEEVENTF\_RIGHTDOWN | MOUSEEVENTF\_RIGHTUP, 0, 0, 0, 0);

            }

            else if (comIN[0] == 0x04)//鼠标向左移动，7

            {

                mouse\_event(MOUSEEVENTF\_MOVE, -moveSpeed, 0, 0, 0);//相对当前鼠标位置x轴分别移动50像素

                Thread.Sleep(moveSleep);//延时30ms

            }

            else if (comIN[0] == 0x01)//鼠标向右移动，3

            {

                mouse\_event(MOUSEEVENTF\_MOVE,  moveSpeed,0, 0, 0);//相对当前鼠标位置x轴分别移动50像素

                Thread.Sleep(moveSleep);//延时30ms

            }

            else if (comIN[0] == 0x07)//鼠标向上移动，6

            {

                mouse\_event(MOUSEEVENTF\_MOVE, 0, -moveSpeed, 0, 0);//相对当前鼠标位置x轴分别移动50像素

                Thread.Sleep(moveSleep);//延时30ms

            }

            else if (comIN[0] == 0x06)//鼠标向下移动，5

            {

                mouse\_event(MOUSEEVENTF\_MOVE, 0, moveSpeed, 0, 0);//相对当前鼠标位置x轴分别移动50像素

                Thread.Sleep(moveSleep);//延时30ms

            }

            else if (comIN[0] == 0x00)//模式切换为键盘，8

            {

                controlMode = 1;

                Thread.Sleep(200);//延时200ms

            }

        }

…………

\*省略部分\*

…………

    }

}

### 14.2 键鼠模拟程序说明（部分）

以下为程序中部分重要变量、线程、函数的简述：

### 变量说明

1. **portName**: 串口名称，默认为"COM5"。
2. **baudRate**: 波特率，默认为9600。
3. **parity**: 校验方式，设置为奇校验。
4. **dataBits**: 数据位，默认为8位。
5. **stopBits**: 停止位，设置为一个停止位。
6. **sp**: 串口对象，用于串口通信。
7. **dataReceiveThread**: 用于接收串口数据的线程对象。
8. **writeReceived**: 用于处理接收到数据的线程对象。
9. **canRecieveMsg**: 标志是否接收数据。
10. **strReceived**: 接收到的字符串数据。
11. **bytesReceived**: 接收到的字节数组。
12. **bytesLength**: 接收到的字节数组长度。
13. **IsOpenSerial**: 标志串口是否打开。
14. **receivedData**: 接收到的数据，默认为空字符串。
15. **moveSpeed**: 鼠标移动速度。
16. **moveSleep**: 鼠标移动延时。
17. **keybdSleep**: 键盘输入延时。
18. **controlMode**: 控制模式，0为鼠标模式，1为键盘模式。

### 线程说明

1. **dataReceiveThread**:
   * 用途: 接收串口数据。
   * 说明: 该线程在后台运行，通过串口对象读取数据，并将数据处理后进行显示或控制鼠标/键盘。
2. **writeReceived**:
   * 用途: 写入接收到的数据。
   * 说明: 该线程在后台运行，用于实时处理和显示接收到的数据。

### 函数说明

1. **Form1()**:
   * 用途: 构造函数，初始化组件和变量。
   * 说明: 初始化窗体组件，设置控件跨线程调用检查为false。
2. **Form1\_Load(object sender, EventArgs e)**:
   * 用途: 窗体加载时执行的函数。
   * 说明: 设置初始串口参数并实例化串口对象。
3. **ClosePort()**:
   * 用途: 关闭串口。
   * 说明: 尝试关闭串口，并终止相关线程。
4. **whileRunning()**:
   * 用途: 持续运行的循环，用于处理串口打开后的操作。
   * 说明: 在串口打开时持续运行，可用于调试或其他后台处理。
5. **connect\_Click(object sender, EventArgs e)**:
   * 用途: 连接按钮点击事件处理。
   * 说明: 打开串口并启动数据接收线程。
6. **textBox1\_TextChanged(object sender, EventArgs e)**:
   * 用途: 文本框内容改变事件处理。
   * 说明: 可用于处理文本框内容变化的逻辑。
7. **DataReceiveFunction()**:
   * 用途: 数据接收函数。
   * 说明: 线程循环读取串口数据，并根据控制模式调用相应的鼠标或键盘控制函数。
8. **clear\_Click(object sender, EventArgs e)**:
   * 用途: 清除按钮点击事件处理。
   * 说明: 清空文本框内容。
9. **DisConnect\_Click(object sender, EventArgs e)**:
   * 用途: 断开连接按钮点击事件处理。
   * 说明: 调用关闭串口函数。
10. **Form1\_FormClosing(object sender, FormClosingEventArgs e)**:
    * 用途: 窗体关闭事件处理。
    * 说明: 在窗体关闭时停止接收数据并关闭串口。
11. **stateTEXT\_TextChanged(object sender, EventArgs e)**:
    * 用途: 状态文本框内容改变事件处理。
    * 说明: 可用于处理状态显示的逻辑。
12. **mousecontroller(byte[] comIN)**:
    * 用途: 控制鼠标操作。
    * 说明: 根据接收到的数据执行鼠标点击或移动操作。
13. **keyboardcontroller(byte[] comIN)**:
    * 用途: 控制键盘操作。
    * 说明: 根据接收到的数据执行键盘按键操作。
14. **nowMODE\_TextChanged(object sender, EventArgs e)**:
    * 用途: 控制模式文本框内容改变事件处理。
    * 说明: 可用于处理模式显示的逻辑。

### 14.3 VERILOG触控板数据处理与发送代码与说明（部分）

**模块概述**

该模块实现了一个基于触摸输入的触控数据处理系统，通过UART发送处理后的触控数据。主要由以下几个部分组成：

1. **触控数据处理模块**：处理8个触摸输入信号，输出一个4位的触控输出信号。
2. **时钟计数器**：用来控制数据发送的时间间隔。
3. **UART发送模块**：将处理后的触控数据通过UART发送出去。

module top\_module (

input wire clk,

input wire reset,

input wire [7:0] touch\_input, // 8个触摸输入引脚

output wire tx // UART输出引脚

);

wire [3:0] touch\_output;

reg [7:0] tx\_data;

reg tx\_data\_valid;

// 实例化触控数据处理模块

touch\_processor processor (

.clk(clk),

.reset(reset),

.touch\_input(touch\_input),

.touch\_output(touch\_output)

);

// 定义23位计数器，用于计数5000000个时钟周期（假设时钟频率为100MHz）

reg [22:0] clock\_counter;

parameter CLOCK\_DIVIDE = 5000000; // 对应50毫秒间隔（100MHz时钟）

// 状态机

always @(posedge clk or posedge reset) begin

if (reset) begin

clock\_counter <= 23'b0;

tx\_data <= 8'b0;

tx\_data\_valid <= 1'b0;

end else begin

if (clock\_counter == CLOCK\_DIVIDE - 1) begin

// 当计数器达到预设值时，重置计数器，并准备发送数据

clock\_counter <= 23'b0;

tx\_data <= {4'b0, touch\_output}; // 将4位触控输出扩展到8位并赋值给tx\_data

tx\_data\_valid <= 1'b1; // 有效标志置位

end else begin

// 否则继续计数

clock\_counter <= clock\_counter + 1;

tx\_data\_valid <= 1'b0; // 无效标志清零

end

end

end

// 实例化UART发送模块

uart\_tx uart (

.clk(clk),

.rst\_n(~reset),

.tx\_data(tx\_data),

.tx\_data\_valid(tx\_data\_valid),

.tx\_pin(tx)

);

Endmodule

**详细注释与说明**

1. **模块输入输出**
   * clk：系统时钟信号。
   * reset：系统复位信号，高电平有效。
   * touch\_input：8个触摸输入信号。
   * tx：UART发送引脚。
2. **内部信号与参数**
   * touch\_output：4位触控输出信号，由触控数据处理模块生成。
   * tx\_data：8位要通过UART发送的数据。
   * tx\_data\_valid：发送数据有效标志位。
   * clock\_counter：23位计数器，用于生成时间间隔信号。
   * CLOCK\_DIVIDE：定义计数器的最大值，对应于100毫秒的时间间隔（假设时钟频率为100MHz）。
3. **触控数据处理模块实例化**
   * touch\_processor：处理8个触摸输入信号，并生成4位触控输出信号。
   * clk：系统时钟信号。
   * reset：系统复位信号。
   * touch\_input：8个触摸输入信号。
   * touch\_output：4位触控输出信号。
4. **状态机**
   * 当reset信号有效时，复位计数器clock\_counter、发送数据tx\_data和数据有效标志位tx\_data\_valid。
   * 在时钟上升沿，如果计数器clock\_counter达到预设值CLOCK\_DIVIDE - 1：
     + 将计数器重置为0。
     + 将4位触控输出信号扩展为8位，并赋值给发送数据tx\_data。
     + 置位发送数据有效标志位tx\_data\_valid。
   * 否则，计数器继续计数，并清零发送数据有效标志位tx\_data\_valid。
5. **UART发送模块实例化**
   * uart\_tx：UART发送模块，用于将数据通过UART发送出去。
   * clk：系统时钟信号。
   * rst\_n：UART复位信号，低电平有效，所以连接~reset。
   * tx\_data：8位要发送的数据。
   * tx\_data\_valid：发送数据有效标志位。
   * tx\_pin：UART发送引脚。
   * 如果没有任何有效的触摸输入信号，则输出无效值4'hF。

### 模块概述

touch\_processor模块用于处理8个触摸输入信号，根据激活的触摸输入信号输出相应的4位序号信号。该模块的主要功能是检测哪个触摸输入被激活，并输出对应的序号。若没有有效触摸输入，则输出无效值。

module touch\_processor (

input wire clk, // 系统时钟信号

input wire reset, // 系统复位信号，高电平有效

input wire [7:0] touch\_input, // 8个触摸输入引脚

output reg [3:0] touch\_output // 输出激活触控信号的序号

);

always @(posedge clk or posedge reset) begin

if (reset) begin

// 当复位信号为高电平时，将触控输出重置为4'h0

touch\_output <= 4'h0;

end else begin

// 根据触摸输入信号的值确定触控输出的值

case (touch\_input)

8'b00000001: touch\_output <= 4'h0; // 触摸输入[0]被激活

8'b00000010: touch\_output <= 4'h1; // 触摸输入[1]被激活

8'b00000100: touch\_output <= 4'h2; // 触摸输入[2]被激活

8'b00001000: touch\_output <= 4'h3; // 触摸输入[3]被激活

8'b00010000: touch\_output <= 4'h4; // 触摸输入[4]被激活

8'b00100000: touch\_output <= 4'h5; // 触摸输入[5]被激活

8'b01000000: touch\_output <= 4'h6; // 触摸输入[6]被激活

8'b10000000: touch\_output <= 4'h7; // 触摸输入[7]被激活

default: touch\_output <= 4'hF; // 没有有效触摸输入，输出无效值4'hF

endcase

end

end

endmodule

**详细注释与说明**

1. **模块输入输出**
   * clk：系统时钟信号。
   * reset：系统复位信号，高电平有效。
   * touch\_input：8个触摸输入信号。
   * touch\_output：4位输出信号，用于表示激活的触控信号的序号。
2. **always 块**
   * 该always块在时钟上升沿(posedge clk)或复位信号上升沿(posedge reset)时触发。
3. **复位条件**
   * 如果reset信号为高电平，触控输出touch\_output被重置为4'h0。
4. **case 语句**
   * 在复位条件之外，根据touch\_input的值确定touch\_output的值。
   * 使用case语句来检测哪个触摸输入信号被激活，并输出相应的序号。
     + 如果touch\_input为8'b00000001，则touch\_output为4'h0。
     + 如果touch\_input为8'b00000010，则touch\_output为4'h1。
     + 如果touch\_input为8'b00000100，则touch\_output为4'h2。
     + 以此类推，直到最高位的触摸输入8'b10000000，对应的输出为4'h7。
   * 如果没有任何有效的触摸输入信号，则输出无效值4'hF。

### 模块概述

uart\_tx模块是一个简单的UART发送器。它将8位并行数据转换为串行数据，并通过UART协议发送。模块支持波特率为9600bps，输入时钟频率为100MHz。它包括四个主要状态：空闲、起始、数据传输和停止。

module uart\_tx (

input wire clk, // 系统时钟信号

input wire rst\_n, // 复位信号，低电平有效

input wire [7:0] tx\_data, // 要发送的8位数据

input wire tx\_data\_valid, // 数据有效信号

output reg tx\_pin // UART发送引脚

);

parameter CLK\_FRE = 100\_000\_000; // 100 MHz时钟频率

parameter BAUD\_RATE = 9600; // 波特率9600

localparam CYCLE = CLK\_FRE / BAUD\_RATE; // 每个波特率周期所需的时钟周期数

// 状态编码

localparam S\_IDLE = 0; // 空闲状态

localparam S\_START = 1; // 起始状态

localparam S\_DATA = 2; // 数据传输状态

localparam S\_STOP = 3; // 停止状态

reg [1:0] state; // 状态寄存器

reg [3:0] bit\_count; // 数据位计数器

reg [15:0] clk\_count; // 时钟计数器

reg [7:0] data; // 数据寄存器

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n) begin

// 复位信号有效时，初始化所有寄存器

state <= S\_IDLE;

bit\_count <= 0;

clk\_count <= 0;

tx\_pin <= 1'b1;

end else begin

case (state)

// 空闲状态

S\_IDLE: begin

if (tx\_data\_valid) begin

state <= S\_START; // 如果数据有效，进入起始状态

data <= tx\_data; // 存储要发送的数据

end

tx\_pin <= 1'b1; // 发送引脚保持高电平

end

// 起始状态

S\_START: begin

if (clk\_count < CYCLE) begin

clk\_count <= clk\_count + 1; // 计数器增加

end else begin

clk\_count <= 0; // 计数器重置

state <= S\_DATA; // 进入数据传输状态

tx\_pin <= 1'b0; // 起始位（低电平）

end

end

// 数据传输状态

S\_DATA: begin

if (clk\_count < CYCLE) begin

clk\_count <= clk\_count + 1; // 计数器增加

end else begin

clk\_count <= 0; // 计数器重置

if (bit\_count < 8) begin

tx\_pin <= data[bit\_count]; // 发送数据位

bit\_count <= bit\_count + 1; // 位计数器增加

end else begin

bit\_count <= 0; // 位计数器重置

state <= S\_STOP; // 进入停止状态

end

end

end

// 停止状态

S\_STOP: begin

if (clk\_count < CYCLE) begin

clk\_count <= clk\_count + 1; // 计数器增加

end else begin

clk\_count <= 0; // 计数器重置

state <= S\_IDLE; // 回到空闲状态

tx\_pin <= 1'b1; // 停止位（高电平）

end

end

endcase

end

end

endmodule

**详细注释与说明**

1. **参数声明**
   * CLK\_FRE: 系统时钟频率，设置为100MHz。
   * BAUD\_RATE: UART波特率，设置为9600bps。
   * CYCLE: 每个波特率周期所需的时钟周期数，用于控制发送数据的速度。
2. **状态声明**
   * S\_IDLE: 空闲状态，等待数据有效信号。
   * S\_START: 起始状态，发送起始位（低电平）。
   * S\_DATA: 数据传输状态，逐位发送8位数据。
   * S\_STOP: 停止状态，发送停止位（高电平）。
3. **寄存器声明**
   * state: 当前状态寄存器。
   * bit\_count: 数据位计数器，用于跟踪已经发送的位数。
   * clk\_count: 时钟计数器，用于生成所需的波特率。
   * data: 数据寄存器，存储要发送的8位数据。
4. **always块**
   * 在时钟上升沿(posedge clk)或复位信号下降沿(negedge rst\_n)时触发。
   * 复位信号有效时，将所有寄存器初始化。
   * 根据当前状态，执行相应操作：
     + 在S\_IDLE状态，如果数据有效信号tx\_data\_valid为高电平，则进入S\_START状态，并将要发送的数据存储在data寄存器中，同时将发送引脚tx\_pin保持高电平。
     + 在S\_START状态，等待一个波特率周期（CYCLE个时钟周期）后，进入S\_DATA状态，并将发送引脚tx\_pin设为低电平，表示起始位。
     + 在S\_DATA状态，逐位发送数据位。每发送一位数据，等待一个波特率周期（CYCLE个时钟周期），直到所有8位数据发送完毕，然后进入S\_STOP状态。
     + 在S\_STOP状态，等待一个波特率周期（CYCLE个时钟周期）后，回到S\_IDLE状态，并将发送引脚tx\_pin设为高电平，表示停止位。