上海大学

SHANGHAI UNIVERSITY

2023-2024 学年夏季学期

课程报告

《计算机硬件综合大型作业报告》

小组序号：1-7

项目名称：六位密码电子锁控制器

指导老师：张云华

组员学号姓名：

组长：22121037 钱鹏程

组员 22121022 王奕憧

22121038 马浩元

22121043 杨铭宇

22121050 胡铭江

计算机工程与科学学院

报告日期 2024 年 6 月 3 日

目录

1 实验项目意义

1.1 现实意义

电子密码锁在安全性方面有着显著的优势。与传统的机械锁相比，电子密码锁通过复杂的数字组合，使得破解难度大大增加。仅仅六位密码就能提供超过一百万种可能的组合，大大提高了安全性，使得盗贼难以通过试错法破解密码。此外，目前市面上流行的电子密码锁通常还具备防暴力破坏的设计，一旦感应到暴力破坏的企图，便会启动报警功能，进一步保障安全。这种高度的安全性使得电子密码锁在家庭、办公室和商业场所中得到了广泛应用，有效保护了个人财产和商业机密。

密码锁的便利性也是其重要的现实意义之一。传统的机械锁依赖于物理钥匙，钥匙的丢失和忘带常常给人们带来不便。而电子密码锁只需记住一组密码即可轻松开锁，无需担心钥匙遗失或被复制。特别是对于家庭成员较多的家庭或人员流动频繁的办公室，电子密码锁的便利性尤为明显。此外，电子密码锁的密码可以随时更改，用户可以根据需要随时更换密码，进一步提高了安全性和灵活性。

智能化是电子密码锁的一大亮点。随着智能家居的普及，电子密码锁逐渐成为智能家居系统的一部分。它可以与其他智能设备联动，实现远程控制和监控。例如，用户可以通过手机 App 远程开锁或查看锁的使用记录，及时了解家中的安全状况。一些高级的电子密码锁甚至可以记录每次开锁的时间和使用的密码，方便用户追踪和管理。这种智能化的功能不仅提高了使用的便捷性，也为用户提供了更全面的安全保障。

从经济角度来看，电子密码锁具有较高的成本效益。虽然初始安装成本相对较高，但其耐用性和低维护性使得长期使用更加经济。相较于传统锁具频繁更换和维修的费用，电子密码锁的使用寿命更长，减少了更换和维修的频率。此外，对于企业和机构而言，电子密码锁可以减少管理物理钥匙的成本和时间，提高运营效率，带来显著的经济效益。

最后，电子密码锁的普及和应用推动了电子技术的发展。电子密码锁的不断改进和升级，不仅提高了自身的安全性和功能性，也促进了相关技术和产业链的进步。例如，随着电子密码锁技术的发展，指纹识别、面部识别等更高级的生物识别技术也逐渐被集成到密码锁中，带来了更多的创新产品和功能。

1.2 学习意义

本项目旨在通过运用大二所学习的《数字逻辑》与《计算机组成原理》的相关知识，实现一个简易的六位密码电子锁控制器。一方面，在实现项目的过程中，我们复习了我们所学的计算机硬件基础知识，能够让我们将学习的理论知识运用于实践当中，反过来还能检验书本上理论的正确性，有利于知识的融会贯通；另一方面，我们初步掌握了专业设计工作的流程和方法，熟练运用计算机模拟软件、版本控制软件等工具提高合作工作效率。

2 实验项目原理

2.1 实验箱组成与原理

本次实验使用的实验箱为 DICE-SEM II 数字模拟综合实验箱，其为一个电子系统综合设计平台（如图 1 所示）。系统组成如下：

电源：交流输入，220V±10%，50H；直流输出，±12V/200mA ；5V/2A。

固定频率脉冲源 7 路：1Hz、10Hz、100Hz、1kHz、10kHz、100kHz、1MHz。

测频范围为 0~300kH 数字式频率计。

两组单脉冲：可同时输出正负两个脉冲，脉冲幅值为 TTL 电平。

四路（T1、T2、T3、T4）时序发生器及启停电路。

逻辑电平的输入与显示：十六位独立逻辑电平开关，可输出“0”、“1”电平（为正逻辑）；十六位由红色和绿色 LED 及驱动电路组成的逻辑电平显示电路。

6 位八段 LED 数码管组成的 BCD 码译码显示器。

Lattice-1032E 所有 I/O 口全部开发，可在线下载我公司提供的集成器件库和用户自己编写的 EDA 程序。并提供相关软件和下载电缆。

其它组成：开发式 IC 插座 40P 一个、20P 三个、16P 两个、8P 一个；多组由锁紧插孔与锁紧小圆孔组合而成的分立元件自助实验区；提供一块小型面包板，以满足不同的需要；两组不同阻值可调电位器。

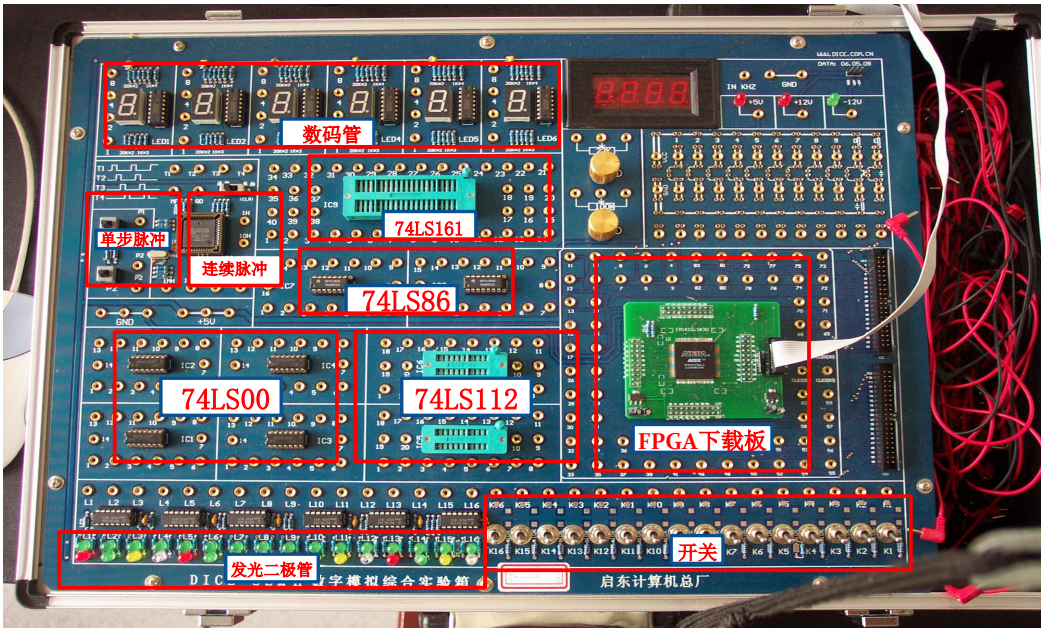


图 1 DICE-SEM 实验箱

以下详细介绍本实验项目所使用的相关模块：时序发生器、八段 LED 数码管、BCD 码译码器、Lattice-1032E。

2.1.1　时序发生器

原理：微程序控制器中使用的时序信号产生器由时钟源、环形脉冲发生器、节拍脉冲和读写时序译码逻辑、启停控制逻辑等部分组成。时钟源用来为环形脉冲发生器提供频率稳定且电平匹配的方波时钟脉冲信号。它通常由石英晶体振荡器和与非门组成的正反馈振荡电路组成，其输出送至环形脉冲发生器。环形脉冲发生器的作用是产生一组有序的间隔相等或不等的脉冲序列，以便通过译码电路来产生最后所需的节拍脉冲。为了在节拍脉冲上不带干扰毛刺，环形脉冲发生器通常采用循环移位寄存器形式。一种典型的环形脉冲发生器及其译码逻辑，它采用循环移位寄存器形式。

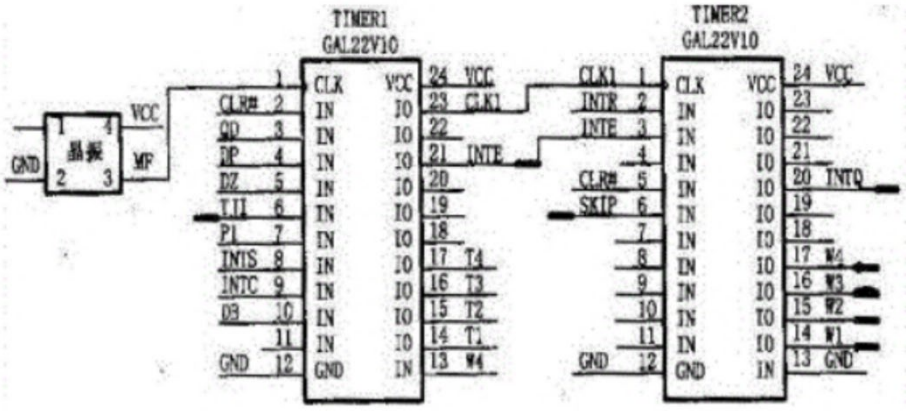


图 2 时序发生器

实验箱中的结构组成：

① 时钟芯片。时钟芯片是时序发生器的核心部件，通常采用晶振或定时器实现。时钟芯片提供稳定的计数脉冲信号，通过调节时钟芯片的参数，可以实现不同的计数频率和计数精度。

② 计数器芯片。计数器芯片是时序发生器的另一个重要部件，通常采用数字集成电路实现。计数器芯片有多个输入引脚和输出引脚，其中包括计数脉冲输入引脚、清零输入引脚、计数输出引脚等。计数器

芯片可以按照不同的计数方案，实现不同的计数方式，例如二进制计数、BCD 码计数等。

③ 逻辑门芯片。逻辑门芯片是时序发生器实现各种时序信号的关键部件。通过将计数器芯片的输出信号与逻辑门芯片的输入信号进行逻辑运算，可以实现各种不同的时序信号的产生。

④ 输出模块。时序发生器的输出信号通常由数码管、LED 灯等模块进行显示或输出，以实现对时序信号的观测和控制。

2.1.2 八段 LED 数码管

LED 数码管是由 8 个发光二极管构成，并按照一定的图形及排列封转在一起的显示器件。其中 7 个 LED 构成 7 笔字形，1 个 LED 构成小数点。

LED 数码管有两大类，一类是共阴极接法，另一类是共阳极接法，共阴极就是 7 段的显示字码共用一个电源的负极，是高电平点亮，共阳极就是 7 段的显示字码共用一个电源的正极，是低电平点亮。只要控制其中各段 LED 的亮灭即可显示相应的数字、字母或符号。

共阴和共阳极数码管的内部电路，它们的发光原理是一样的，只是它们的电源极性不同而已，共阴为所有的 LED 负极接在一起，共阳为所有的 LED 正极接在一起。如图 3 为 1 位数码管的共阴极和共阳极原理图及其电路图：

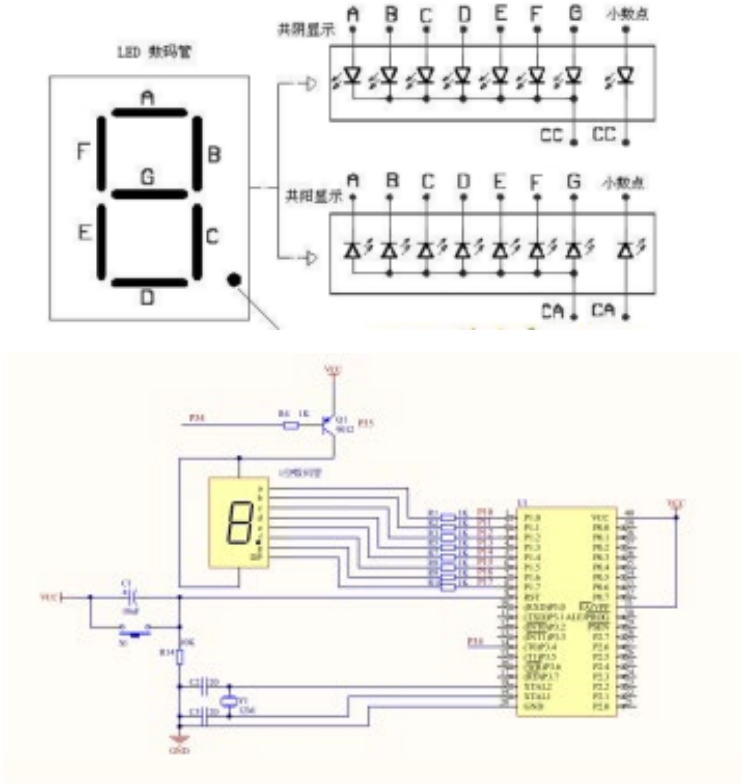


图 3 八段 LED 数码管电路图

对数码管所要显示的每个数字和字母进行编码，然后在编程时，将编码放在一个数组上，需要显示什么数字或者字母，从数组里面提取相应的编码就可显示所要显示的字符，同时，根据 LED 数码管的驱动方式的不同，可以分为静态式和动态式两类：

①静态驱动方式。静态驱动也称直流驱动。静态驱动是指每个数码管的每一个段码都由一个单片机的 I/O 口进行驱动，或者使用如 BCD 码二-十进位转换器进行驱动。静态驱动的优点是编程简单，显示亮度高，缺点是占用 I/O 口多

②动态驱动方式。数码管动态显示介面是单片机中应用最为广泛的一种显示方式之一，动态驱动是将所有数码管的 8 个显示笔划"a, b, c, d, e, f, g, dp "的同名端连在一起，另外为每个数码管的公共极 COM 增加位选通控制电路，位选通由各自独立的 I/O 线控制。

当单片机输出字形码时，所有数码管都接收到相同的字形码，但究竟是那个数码管会显示出字形，取决于

单片机对位元选通 COM 端电路的控制，所以我们只要将需要显示的数码管的选通控制打开，该位就显示出字形，没有选通的数码管就不会亮。

透过分时轮流控制各个 LED 数码管的 COM 端，就使各个数码管轮流受控显示，这就是动态驱动。在轮流显示过程中，每位数码管的点亮时间为 1~2ms，由于人的视觉暂留现象及发光二极体的余辉效应，尽管实际上各位数码管并非同时点亮，但只要扫描的速度足够快，给人的印象就是一组稳定的显示资料，不会有闪烁感，动态显示的效果和静态显示是一样的，能够节省大量的 I/O 口，而且功耗更低。

2.1.3 BCD 码译码器

分段式显示器（LED 数码管）由 7 条线段围成 8 型，每一段包含一个发光二极管。外加正向电压时二极管导通，发出清晰的光，有红、黄、绿等色。只要按规律控制各发光段的亮、灭，就可以显示各种字形或符号。

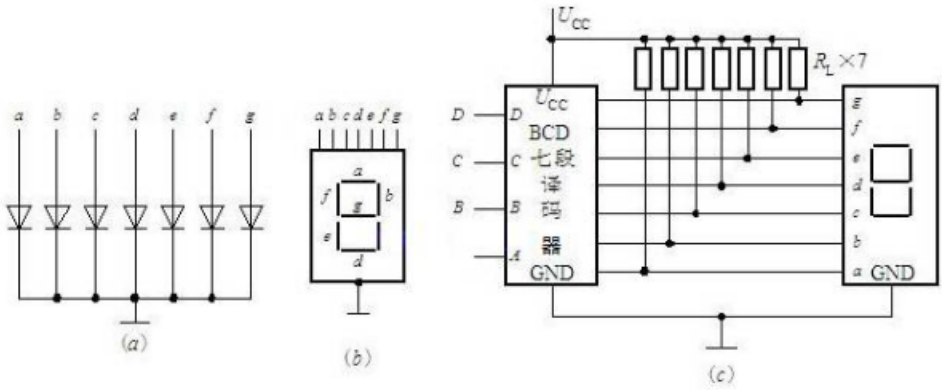


图 4 5BCD 七段数码管译码器

图 4 中，图(a)是共阴式 LED 数码管的原理图，使用时，公共阴极接地，7 个阳极 a~g 由相应的 BCD 七段译码器来驱动（控制），如图(c)以及图 6 所示。

BCD 七段译码器的输入是一位 BCD 码（以 D、C、B、A 表示），输出是数码管各段的驱动信号(以 Fa~Fg 表示)，也称 4-7 译码器。若用它驱动共阴 LED 数码管，则输出应为高有效，即输出为高 1 时，相应显示段发光。例如，当输入 8421 码 DCBA=0100 时，应显示 4，即要求同时点亮 b、c、f、g 段，熄灭 a、d、e 段，故译码器的输出应为 F_a~F_g=0110011，这也是一组代码，常称为段码。同理，根据组成 0~9 这 10 个字形的要求可以列出 8421BCD 七段译码器的真值表。

输入				输出							字形
D	C	B	A	F_a	F_b	F_c	F_d	F_e	F_f	F_g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9

图 5 5BCD 七段数码管真值表

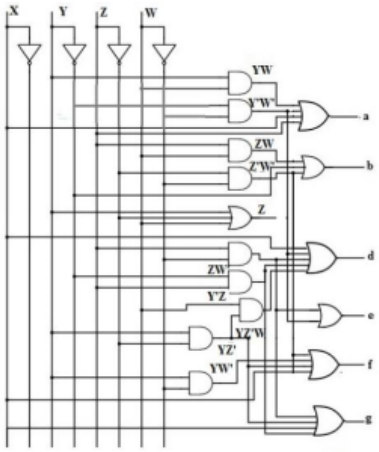


图 6 5BCD 七段数码管译码功能电路图

2.1.4 Lattice-1032E

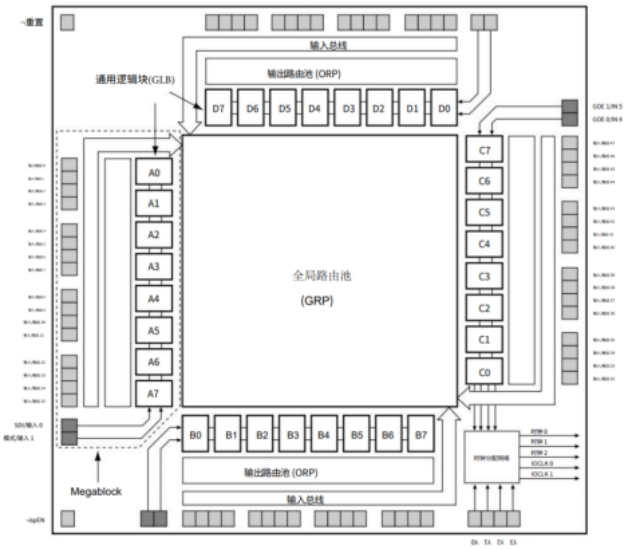


图 7 Lattice-1032E 芯片

Lattice ispLSI 1032E 芯片是一款高密度可编程逻辑器件，包含 192 个寄存器、64 个通用 I/O 引脚、8 个专用输入引脚、4 专用时钟输入引脚和一个全局路由池(GRP)。GRP 提供所有这些元素之间的完整互连。ispLSI 1032E 器件提供逻辑的 5V 非易失性系统内可编程性，以及提供真正可重新配置系统的互连。

ispLSI 1032E 器件上的基本逻辑单元是通用逻辑块(GLB)。GLB 标记为 A0、A1…D7（见图 6）。ispLSI 1032E 器件中共有 32 个 GLB。每个 GLB 有 18 个输入、1 个可编程的与/或/异或阵列，以及 4 个可配置为组合或寄存的输出。GLB 的输入来自 GRP 和专用输入。所有 GLB 输出都被带回 GRP，以便它们可以连接到设备上任何 GLB 的输入。

该器件还有 64 个 I/O 单元，每个单元都直接连接到一个 I/O 引脚。每个 I/O 单元均可单独编程为组合输入、寄存输入、锁存输入、输出或具有三态控制的双向 I/O 引脚。信号电平是 TTL 兼容电压，输出驱动器可以提供 4 mA 电流或吸收 8 mA 电流。每个输出均可针对快速或慢速输出转换率独立编程，以最大限度地降低总体输出开关噪声。

八个 GLB、16 个 I/O 单元、两个专用输入和一个 ORP 连接在一起构成一个 Megablock（如图 7）。八个 GLB 的输出通过 ORP 连接到一组 16 个通用 I/O 单元。每个 ispLSI 1032E 设备包含四个 Megablocks。

GRP 将所有 GLB 的输出和双向 I/O 单元的所有输入作为其输入。所有这些信号都可用于 GLB 的输入。通过 GRP 的延迟已被均衡以最小化时序偏差。

ispLSI 1032E 器件中的时钟是使用时钟分配网络选择的。四个专用时钟引脚（Y0、Y1、Y2 和 Y3）被引入分配网络，并提供五个时钟输出（CLK 0、CLK 1、CLK 2、IOCLK 0 和 IOCLK 1）以将时钟路由到 GLB 和 I/O 细胞。时钟分配网络也可以由特殊时钟 GLB（ispLSI 1032E 设备上的 C0）驱动。该 GLB 的逻辑允许用户根据设备内部信号的组合创建内部时钟。

2.2 Quartus II 软件

Altera Quartus II 是一种可编程逻辑器件电子设计自动化开发软件，也叫做 FPGA 开发环境。它可以识别电路的 Verilog 或 VHDL 高级硬件描述语言表述，或读取指定格式的线路图；进而完成逻辑仿真、功能验证、逻辑综合等任务，对器件的进行编程，即将设计项目转换到实际的硬件。该软件提供了逻辑电路的可视化设计以及向量波形的仿真等功能。

通过使用该软件，经过一系列操作即可将本实验项目编写的 Verilog HDL 语言文件转换成可视化的电路图、生成模拟的仿真波形文件、以及下载到现实的 FPGA 硬件中进行测试。

2.3 Verilog HDL 编程语言

Verilog HDL（简称 Verilog）是一种硬件描述语言，用于数字电路的系统设计。可对算法级、门级、开关级等多种抽象设计层次进行建模。Verilog 继承了 C 语言的多种操作符和结构，与另一种硬件描述语言 VHDL 相比，语法不是很严格，代码更加简洁，更容易上手。Verilog 不仅定义了语法，还对语法结构都定义了清晰的仿真语义。因此 Verilog 编写的数字模型就能够使用 Verilog 仿真器进行验证。Verilog 具有很强的电路描述与建模能力，能从多个层次对数字系统进行描述和建模。因此，在简化硬件设计任务、提高设计效率与可靠性、语言易读性、层次化和结构化设计等方面展现了强大的生命力与潜力。

下面是一些本实验项目涉及的相关语法的简单介绍。

2.3.1 设计方法

Verilog 的设计多采用自上而下的设计方法(top-down)（如图 8）。即先定义顶层模块功能，进而分析要构成顶层模块的必要子模块；然后进一步对各个模块进行分解、设计，直到到达无法进一步分解的底层功能块。这样，可以把一个较大的系统，细化成多个小系统，从时间、工作量上分配给更多的人员去设计，从而提高了设计速度，缩短了开发周期。

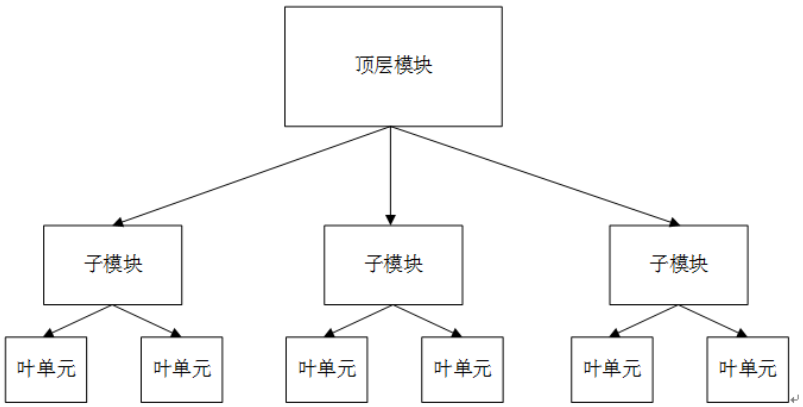


图 8 Verilog 设计整体框架图

2.3.2 基本数据类型

包括 wire（导线）、reg（寄存器）、向量。

在 Verilog HDL 中，wire 表示导线类型。它可以用来描述模块内部信号的连接，并在模块中进行操作。wire 类型的信号只能被连接到模块输入或输出端口，不能被赋值。在模块中，wire 类型的信号通常用于实现组合逻辑电路，用于连接输入、输出和内部逻辑单元。

reg 表示寄存器类型。它=用来表示存储单元，它会保持数据原有的值，直到被改写。在模块中，reg 类型。在 always 块中，寄存器可能被综合成边沿触发器，在组合逻辑中可能被综合成 wire 型变量。寄存器不需要驱动源，也不一定需要时钟信号。在仿真时，寄存器的值可在任意时刻通过赋值操作进行改写。

当位宽大于 1 时，wire 或 reg 即可声明为向量的形式。对于向量，我们可以指定某一位或若干相邻位，

作为其他逻辑使用。

2.3.3　关键字 assign

在 Verilog HDL 中，assign 是一种关键字，用于连接信号或计算信号的值。它可以用来实现组合逻辑电路，将多个输入信号或逻辑单元的输出信号连接到一个输出端口上，从而产生一个输出信号。assign 关键字用于声明一个连续赋值语句，使用等式的形式将某个信号或表达式赋值给另一个信号。assign 语句通常放在模块的最外层，用于连接模块输入和输出端口，或者连接内部逻辑单元的输出信号。

2.3.4　always 语句

always 块是 Verilog 中最常用的一个语法点，always 块是 Verilog 中用来描述组合逻辑以及时序逻辑的语法。这些硬件块都是相互独立同时工作的。每个块之间的连接是决定数据流的原因。为了模拟这种行为，一个 always 块被做成一个连续的过程（硬件不可能断断续续工作），当敏感列表中的一个信号变化时，它就会被触发并执行一些动作（always 块内的语句）。

3　实验项目设计

本项目基于前文介绍过的 Quartus II 6.0 软件以及 Verilog HDL 语言编写，并已经过 DICE SEM II 实验箱的上机调试和验收。该项目各主要文件的说明如下：

```
├── error_counter.v                          // 错误计数器模块
├── judge.v                                  // 密码正确与否的判断模块
├── led_flasher.v                            // 报警闪烁灯控制模块
├── passwd_register.v                        // 密码寄存器模块
├── register.v                               // 底部寄存器模块
├── SixDigit_Electronic_Lock_Controller.v    // 顶层电路设计模块
├── yima2to4.v                               // 2-4 译码器模块
├── SixDigit_Electronic_Lock_Controller.qpf  // 项目主文件
├── f1.vmf                                   // 情景一仿真模拟波形文件
├── f2.vmf                                   // 情景二仿真模拟波形文件
├── f3.vmf                                   // 情景三仿真模拟波形文件
├── f4.vmf                                   // 情景四仿真模拟波形文件
├── f5.vmf                                   // 情景五仿真模拟波形文件
                                             // 其他文件略
```

下面按照主模块中调用的顺序来介绍各元件模块以及介绍顶层主模块设计。

3.1 元件设计

3.1.1　2-4 译码器模块

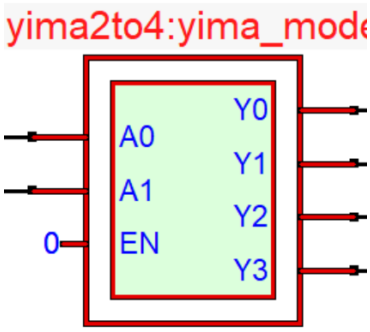


图 9 2-4 译码器

A0	A1	Y0	Y1	Y2	Y3

0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

表 1 2-4 译码器真值表

该模块主要用于：

① 输入密码位数的切换以及输入密码判断的开启。由于项目要求为六位十进制的密码，故每一位密码需要 4 个开关输入，考虑到实验箱提供的开关有限，故本项目采用每两位密码输入一次的形式。在该功能中，Y0 表示前两位密码的输入，Y1 表示中两位密码的输入，Y2 表示后两位密码的输入，Y3 表示判断模块的开启。

② 多密码选择的功能。本功能为本小组设计的拓展功能。Y0、Y1、Y2、Y3 分别代表多密码中的第一、二、三、四个密码。

该模块代码如下：

```
// 24 译码器
module yima2to4(A0, A1, EN, Y0, Y1, Y2, Y3);
  input A0, A1, EN;
  output reg Y0, Y1, Y2, Y3;

  always @(*) begin
      if (EN) begin
          {Y3, Y2, Y1, Y0} = 4'b0000;
      end else begin
          case({A0, A1})
              2'b00: {Y3, Y2, Y1, Y0} = 4'b0001;
              2'b01: {Y3, Y2, Y1, Y0} = 4'b0010;
              2'b10: {Y3, Y2, Y1, Y0} = 4'b0100;
              2'b11: {Y3, Y2, Y1, Y0} = 4'b1000;
              default: {Y3, Y2, Y1, Y0} = 4'b0000;
          endcase
      end
  end

endmodule
```

3.1.2 密码寄存器模块

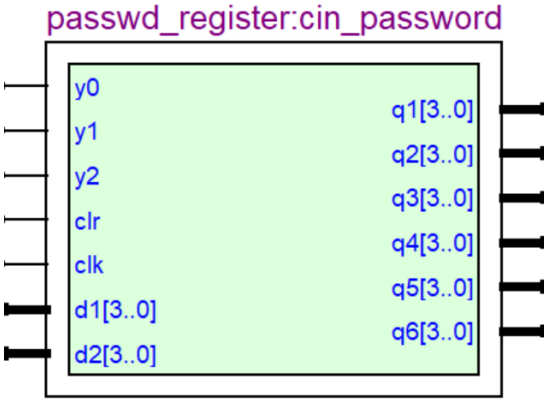


图 10 密码寄存器模块

该模块主要用于存储输入的密码和输出的密码。其中：y0、y1、y2 为从译码器接受到的传入信号；clr 为清空信号，可以清空寄存器中的数据；clk 为时钟信号，请注意此处的时钟信号并非连续的脉冲，而是为方便控制的手动控制的单步脉冲；d1 和 d2 即为输入的两位密码；q1-q6 即为保存好的密码。当传入的密码为非十进制数时，则会在主模块经过处理后传入 E（error，错误），当检测到 E 时保存的密码为 0。

该模块代码如下：

```
//密码寄存器组
module passwd_register(d1,d2,y0,y1,y2,q1,q2,q3,q4,q5,q6,clr,clk);

  input clk,clr;
  input [3:0] d1,d2;
  input y0,y1,y2;
  output [3:0] q1,q2,q3,q4,q5,q6;

  wire [3:0] d1_checked, d2_checked;

  // 非法传入（E）检查
  assign d1_checked = (d1 == 4'b1110) ? 4'b0000 : d1;
  assign d2_checked = (d2 == 4'b1110) ? 4'b0000 : d2;

  // 高两位
  register r1(d1_checked[3:0],q1[3:0],clk&y0,clr);
  register r2(d2_checked[3:0],q2[3:0],clk&y0,clr);

  // 中两位
  register r3(d1_checked[3:0],q3[3:0],clk&y1,clr);
  register r4(d2_checked[3:0],q4[3:0],clk&y1,clr);

  // 低两位
  register r5(d1_checked[3:0],q5[3:0],clk&y2,clr);
  register r6(d2_checked[3:0],q6[3:0],clk&y2,clr);

endmodule
```

该模块本质上是一些预处理加上多个底层寄存器组成的模块，底层寄存器模块代码如下：

```
// 寄存器元单元，时钟上升沿打入数据
module register(d,q,clk,clr);
  input[3:0] d;
  output reg [3:0] q;
  input clk,clr;
  always@(posedge clk or posedge clr)
      begin
          if(clr)
              q <= 4'b0000;
          else if (d != 4'b1110)
              q <= d;
```

```
        end
    endmodule
```

3.1.3 比较密码模块

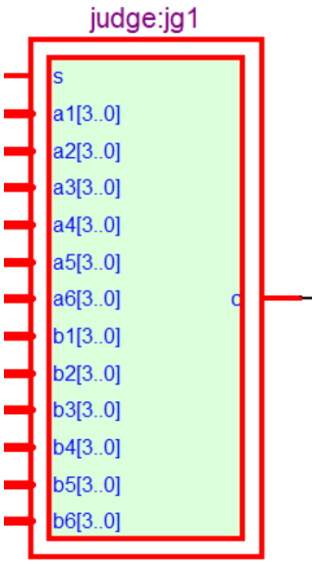


图 11 比较密码模块

该模块主要用于将设置和输入的密码进行比较判断，当 s 开启且 a1-a6 与 b1-b6 完全一致时才会得到判断正确的结果。其中，s 为启动判断的开关，a1-a6 以及 b1-b6 分别为设置好的密码和输入的密码，c 为判断结果（正确 1，错误 0）。

该模块代码如下：

```
// 判断模块
module judge(
    input s,
    input [3:0] a1, a2, a3, a4, a5, a6,
    input [3:0] b1, b2, b3, b4, b5, b6,
    output reg c
);

    always @(s) begin
        if (s) begin
            if (a1 == b1 && a2 == b2 && a3 == b3 && a4 == b4 && a5 == b5 && a6 == b6)
begin
                c <= 1'b1;
            end else begin
                c <= 1'b0;
            end
        end else begin
            c <= 1'b0;
        end
    end

endmodule
```

3.1.4 错误计数器模块

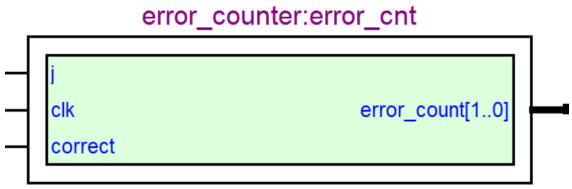


图 12 错误计数器模块

该模块主要用于记录错误次数，可进行从 00~11 的计数，当 11 再次计数时会变为 01，只有当不再错误计数才会变成 00，为项目要求中的连续错三次进行灯闪烁十秒做准备。其中，j 为译码器传过来的 y3，clk 仍为手动时钟信号，correct 为判断模块传过来的判断结果，error_count 为错误计数。

该模块代码如下：

```
// 错误计数器，记录错误次数
module error_counter(
  input j,    // 传入判断
  input clk,
   input correct,
   output reg [1:0] error_count
);
   always @(posedge clk) begin
     if (j) begin
         if (correct) begin
             error_count <= 2'b00;
         end else if (error_count <= 2'b11) begin
         if (error_count == 2'b00) begin
                 error_count <= 2'b01;
             end else if (error_count == 2'b01) begin
                 error_count <= 2'b10;
             end else if (error_count == 2'b10) begin
                 error_count <= 2'b11;
             end else if (error_count == 2'b11) begin
                 error_count <= 2'b01;
             end
         end
     end
  end
endmodule
```

3.1.5 LED 闪烁控制模块

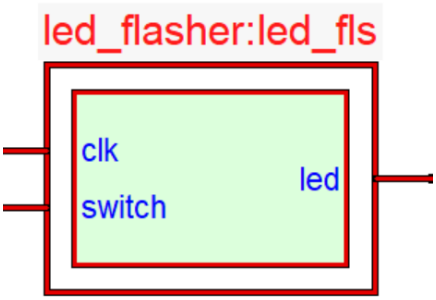


图 13 LED 闪烁控制模块

该模块主要用于 LED 灯的闪烁控制。其中，clk 与之前模块中所用的 clk 不一样，是自动的连续脉冲而非手动的但不脉冲；switch 是错误计数器传来的 error_count 相与的结果，也是打开 led 闪烁的开关，led 即为输出信号。

该模块代码如下：

```
// 控制灯的闪烁
module led_flasher (
    input wire clk,              // 时钟信号，10Hz
    input wire switch,           // 开关信号
    output reg led           // LED 输出
);
    reg [3:0] second_count;  // 计时寄存器
    reg blinking;           // 标志寄存器，指示是否正在闪烁
  reg switch_prev;         // 记录上一个时钟周期的开关状态

    always @(posedge clk) begin
      switch_prev <= switch;

       if (!switch_prev && switch) begin
       // 开关从关闭到打开的瞬间，初始化计数器和标志
          blinking <= 1;
          second_count <= 0;
          led <= 0;
      end

      if (blinking) begin
          if (second_count < 15) begin
          // 计时进行中，LED 跟随时钟闪烁
              second_count <= second_count + 1;
              led <= ~led;
          end else begin
          // 计时结束，停止闪烁
              blinking <= 0;
              led <= 0;
          end

      end else begin
```

```
        // 如果没有在闪烁，则保持 LED 关闭
            led <= 0;
        end
    end
endmodule
```

3.2 电路设计

电路图详见附录。以下详细介绍各接口：

	接口名	说明
输入	m	切换模式，接开关，0 为设置密码，1 为输入密码
	[3:0] inA, inB	两个十进制位的密码输入，接 6 个开关，非 10 进制会不显示且传入 0 到寄存器
	clk	手动信号，接一个单步脉冲，按一下输入（设置）一次密码或进行一次判断
	clr	清空，接开关，同时清空数码管和当前模式对应的寄存器
	true_clk	连续的时钟信号，接 10Hz 口，主要用于控制报警灯的闪烁
	a0, a1	切换高两位、中两位、低两位密码以及判断，接 2 个开关，是实验箱开关数量有限的无奈之举
	ps0, ps1	多密码的切换，接 2 个开关
	hide	输入时隐藏密码（类似于软件输入形如“******”的密码样式），接一个开关
输出	out1-6	显示密码，分别接数码管对应的 24 个接口
	res	显示判断结果，接 1 个 led 灯，当且仅当 a0a1=11 且按下 clk 时才可能发生改变
	eo	错误计数，接 2 个 led 灯
	led	报警灯，接 1 个 led 灯，输入错误三次时自动开始报警闪烁

表 2 总模块各接口介绍

顶层主模块代码如下：

```
module SixDigit_Electronic_Lock_Controller(
    input m,              // mode，切换模式，0 为设置密码，1 为输入密码
    input [3:0] inA, inB, // 输入
    input clk, clr, true_clk,       // 手动时钟信号，清空，自动时钟信号
    input a0, a1,      // 高、中、低位输入与判断选择
    input ps0,ps1,    // 多密码选择
    input hide,       // 隐藏密码
    output [3:0] out1, out2, out3, out4, out5, out6, // 输出
    output res,        // 比较结果
  output led,       // LED 输出
  output [1:0] eo        // 错误计数器的输出
);

    // 内部信号
  wire [3:0] set1_out1, set1_out2, set1_out3, set1_out4, set1_out5, set1_out6;
```

```
 wire [3:0] set2_out1, set2_out2, set2_out3, set2_out4, set2_out5, set2_out6;
 wire [3:0] set3_out1, set3_out2, set3_out3, set3_out4, set3_out5, set3_out6;
 wire [3:0] set4_out1, set4_out2, set4_out3, set4_out4, set4_out5, set4_out6;
 wire [3:0] cin_out1, cin_out2, cin_out3, cin_out4, cin_out5, cin_out6;
 wire [3:0] in1, in2, in3, in4, in5, in6, checked_inA, checked_inB;
 wire y0, y1, y2, y3;      // 高位输入、中位输入、低位输入、判断选择
 wire p0, p1, p2, p3;    // 密码 1、密码 2、密码 3、密码 4
wire res1,res2,res3,res4; // 每个密码的比较结果
wire [1:0] error_count;
wire res_tmp;
 reg [3:0] out1_reg, out2_reg, out3_reg, out4_reg, out5_reg, out6_reg;
reg start_flashing;
reg res_reg;

assign res_tmp = (res1|res2|res3|res4) & m;

// 如果输入非进制数则显示 E(error)
assign checked_inA = (inA > 4'b1001) ? 4'b1110 : inA;
assign checked_inB = (inB > 4'b1001) ? 4'b1110 : inB;

  // 输出赋值
  assign out1 = out1_reg;
  assign out2 = out2_reg;
  assign out3 = out3_reg;
  assign out4 = out4_reg;
  assign out5 = out5_reg;
  assign out6 = out6_reg;
assign res = res_reg;
assign eo = error_count;

  // 模式选择 24 译码器实例化
  yima2to4 yima_mode(a0, a1, 0, y0, y1, y2, y3);

  // 多密码选择 24 译码器实例化
  yima2to4 yima_passwd(ps0, ps1, 0, p0, p1, p2, p3);

  // 设置密码寄存器实例化
  passwd_register set_password1(
      checked_inA, checked_inB, (!m&p0) & y0, (!m&p0) & y1, (!m&p0) & y2,
      set1_out1, set1_out2, set1_out3, set1_out4, set1_out5, set1_out6,
      clr & (!m&p0), clk
  );

passwd_register set_password2(
      checked_inA, checked_inB, (!m&p1) & y0, (!m&p1)& y1, (!m&p1) & y2,
```

```
        set2_out1, set2_out2, set2_out3, set2_out4, set2_out5, set2_out6,
        clr & (!m&p1), clk
    );

    passwd_register set_password3(
        checked_inA, checked_inB, (!m&p2) & y0, (!m&p2) & y1, (!m&p2) & y2,
        set3_out1, set3_out2, set3_out3, set3_out4, set3_out5, set3_out6,
        clr & (!m&p2), clk
    );

    // (!m) & y0
    passwd_register set_password4(
        checked_inA, checked_inB, (!m&p3) & y0, (!m&p3) & y1, (!m&p3) & y2,
        set4_out1, set4_out2, set4_out3, set4_out4, set4_out5, set4_out6,
        clr & (!m&p3), clk
    );

    // 输入密码寄存器实例化
    passwd_register cin_password(
        checked_inA, checked_inB, m & y0, m & y1, m & y2,
        cin_out1, cin_out2, cin_out3, cin_out4, cin_out5, cin_out6,
        clr & (m), clk
    );

    // 比较模块实例化
    judge jg1(y3&m, cin_out1, cin_out2, cin_out3, cin_out4, cin_out5, cin_out6,
set1_out1, set1_out2, set1_out3, set1_out4, set1_out5, set1_out6, res1);
    judge jg2(y3&m, cin_out1, cin_out2, cin_out3, cin_out4, cin_out5, cin_out6,
set2_out1, set2_out2, set2_out3, set2_out4, set2_out5, set2_out6, res2);
    judge jg3(y3&m, cin_out1, cin_out2, cin_out3, cin_out4, cin_out5, cin_out6,
set3_out1, set3_out2, set3_out3, set3_out4, set3_out5, set3_out6, res3);
    judge jg4(y3&m, cin_out1, cin_out2, cin_out3, cin_out4, cin_out5, cin_out6,
set4_out1, set4_out2, set4_out3, set4_out4, set4_out5, set4_out6, res4);

    // 错误计数器模块实例化
    error_counter error_cnt(y3&m, clk, res_tmp, error_count);

    // LED 闪烁控制状态机模块实例化
    led_flasher led_fls(true_clk, start_flashing, led);

    always @(posedge true_clk) begin
    // 错误统计
        if (error_count == 2'b11) begin
            start_flashing <= 1;
        end else begin
```

```
            start_flashing <= 0;
        end
    end

    // 模式选择和输出赋值的手动控制逻辑
    always @(posedge clr or posedge clk) begin
        if (clr) begin
            out1_reg <= 4'b0000;
            out2_reg <= 4'b0000;
            out3_reg <= 4'b0000;
            out4_reg <= 4'b0000;
            out5_reg <= 4'b0000;
            out6_reg <= 4'b0000;
        end

            else begin
                    if (error_flag) begin
                    out1_reg <= 4'b1110;
                    out2_reg <= 4'b1110;
                    out3_reg <= 4'b1110;
                    out4_reg <= 4'b1110;
                    out5_reg <= 4'b1110;
                    out6_reg <= 4'b1110;
                end

            case ({y2, y1, y0})
                3'b001: {out1_reg, out2_reg} <= {checked_inA, checked_inB};
                3'b010: {out3_reg, out4_reg} <= {checked_inA, checked_inB};
                3'b100: {out5_reg, out6_reg} <= {checked_inA, checked_inB};
            endcase

            if (y3) begin
                res_reg <= res_tmp;
            end
        end
    end
endmodule
```

4 实验项目调研过程

4.1 仿真模拟

4.1.1 情景一：单密码输入正确

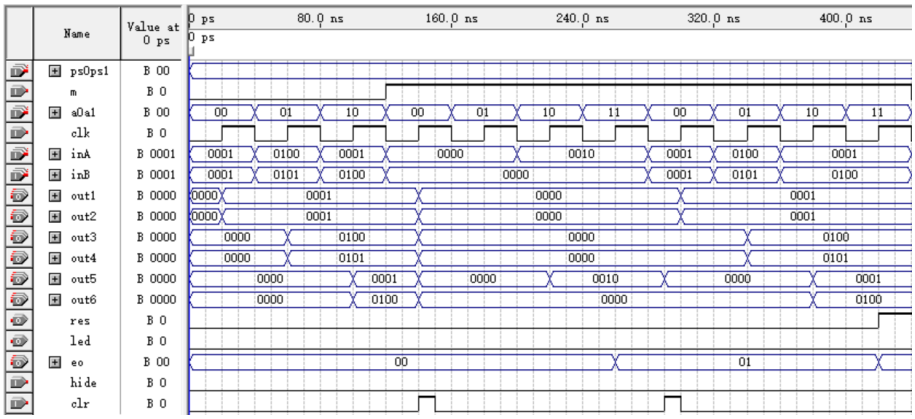


图 14 情景一仿真模拟

如图 14 所示：

① 首先在设置密码阶段（m=0），依次输入六位十进制密码 114514（a0、a1 以及 inA、inB），数码管 out1-6 显示正常；

② 接着切换模式到输入密码阶段（m=1），清空后依次输入六位十进制密码 000020，数码管显示正常，开启判断（a0a1=11），错误计数器（eo）加一，判断结果错误（res=0）；

③ 然后清空后再次输入六位十进制密码 114514，开启判断，错误计数器清零，判断结果正确（res=0）。

4.1.2 情景二：密码复位

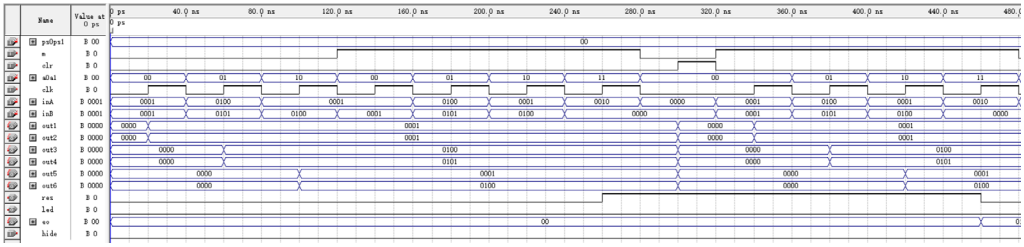


图 15 情景二仿真模拟

如图 15 所示：

① 在情景一的基础之上，切回了设置密码模式，使用了复位密码（clr=1），

② 再切换到输入密码模式，输入密码 114514，与复位后的密码（000000）进行比较，判断结果错误。

4.1.3 情景三：密码输入连续错误三次

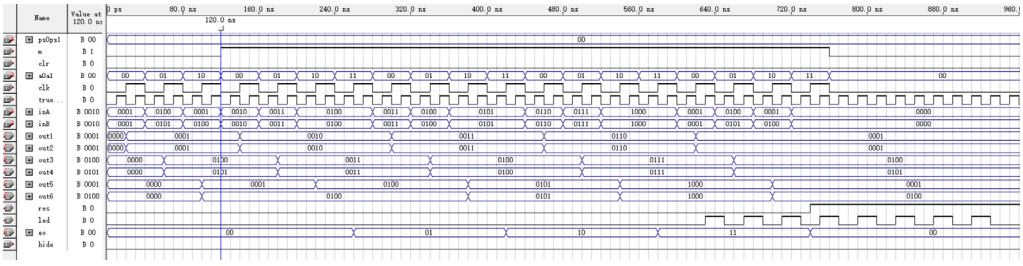


图 16 情景三仿真模拟

如图 16 所示：

① 首先设置密码 114514，输入密码 223344，判断，错误计数器加一；

② 输入密码 334455，判断，错误计数器加一；

③ 输入密码 667788，判断，错误计数器加一，警报指示灯开始闪烁（led）；

④ 输入密码 114514，判断，错误计数器清零。请注意：此处为了仿真文件的方便，在错误计数器清零之后警报指示灯仍在闪烁，实际测试时不冲突。

4.1.4 情景四：多密码输入正确*

本功能为拓展功能，灵感来源于手机解锁可以设置多个指纹或多个人脸方式，在本项目中具体体现为

可以至多设置 4 个密码（通过修改译码器可以设置更多），而输入密码只需要有能与设置的所有密码中某一个相匹配即判断正确。

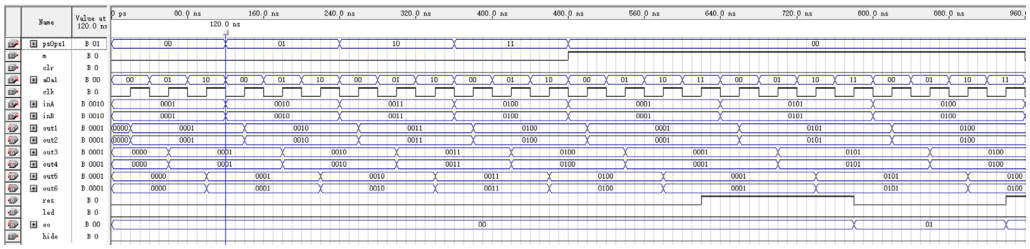


图 17 情景四仿真模拟

如图 17 所示：

① 首先设置第一个密码为 111111，第二个密码为 222222，第三个密码为 333333，第四个密码为 444444；

② 输入密码 111111，判断正确；

③ 输入密码 555555，判断错误；

④ 输入密码 444444，判断正确。

4.1.5 情景五：隐藏密码功能*

本功能为拓展功能，灵感来源于数码设备输入密码时的自动隐藏（形如“******”）功能，在本项目中具体体现为开启隐藏功能后，只有当前输入的两位会显示，其他四位被隐藏掉，具体实现是将其他几位的数码管输出 1110，因为大于 10 数码管不显示。

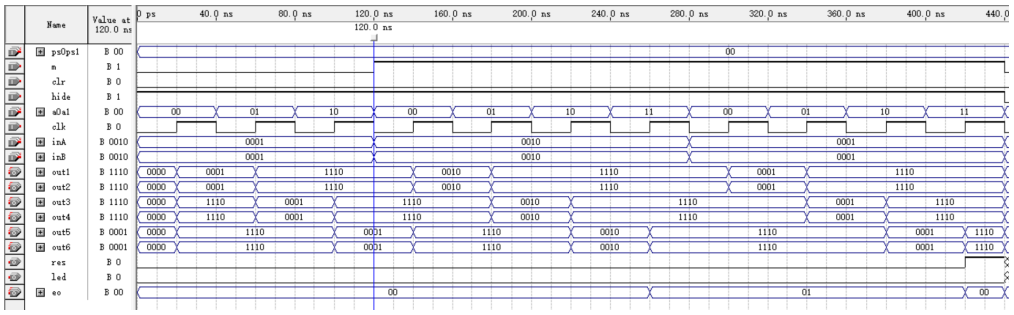


图 18 情景五仿真模拟

如图 18 所示：只要在隐藏开关开启时（hide=1），只有当前输入的两位为输入的数，而其他两位则会是 1110，在此时项目其他各种功能均可正常使用。

4.2 上机调试

4.2.1 引脚设置

首先在 Quartus II 6.0 软件中打开 Pins 设置好如下节点名称与引脚位置的关系：

Node Name	Location	Node Name	Location	Node Name	Location
a0	PIN_138	m	PIN_38	out4[2]	PIN_26
a1	PIN_137	out1[0]	PIN_8	out4[3]	PIN_81
clk	PIN_119	out1[1]	PIN_9	out5[0]	PIN_80
clr	PIN_118	out1[2]	PIN_13	out5[1]	PIN_79
eo[0]	PIN_30	out1[3]	PIN_17	out5[2]	PIN_78
eo[1]	PIN_29	out2[0]	PIN_96	out5[3]	PIN_73
inA[0]	PIN_136	out2[1]	PIN_95	out6[0]	PIN_62
inA[1]	PIN_135	out2[2]	PIN_92	out6[1]	PIN_63
inA[2]	PIN_133	out2[3]	PIN_91	out6[2]	PIN_64
inA[3]	PIN_133	out3[0]	PIN_90	out6[3]	PIN_65
inB[0]	PIN_131	out3[1]	PIN_19	ps0	PIN_69
inB[1]	PIN_130	out3[2]	PIN_88	ps1	PIN_70

inB[2]	PIN_121	out3[3]	PIN_87	res	PIN_68
inB[3]	PIN_120	out4[0]	PIN_86	true_clk	PIN_67
led	PIN_37	out4[1]	PIN_83	hide	PIN_21

表 3 软件内引脚设置

再根据实验箱引脚对照表（如图 14）接线。

实验箱 ISP1032 引脚	MAX+plus II 软件 EP1K10 引脚	实验箱 ISP1032 引脚	MAX+plus II 软件 EP1K10 引脚
3	131	47	59
4	130	48	60
5	133	49	62
6	132	50	63
7	136	51	64
8	135	52	65
9	138	53	68
10	137	54	67
11	8	55	70
12	9	56	69
13	13	57	73
14	17	58	72
15	18	59	79
16	19	60	78
17	21	CLK0（20）	126
18	23	CLK1（61）	125
26	26	CLK2（63）	55
27	27	CLK3（62）	54
28	29	68	81
29	30	69	80
30	32	70	86
31	33	71	83
32	36	72	88
33	37	73	87
34	38	74	91
35	39	75	90
36	41	76	95
37	43	77	92
38	44	78	117
39	46	79	96
40	47	80	119
41	48	81	118
45	49	82	121
46	51	83	120

表 4 实验箱引脚对照表

4.2.2 线路连接、

根据表 3、表 4 链接线路，如下图：

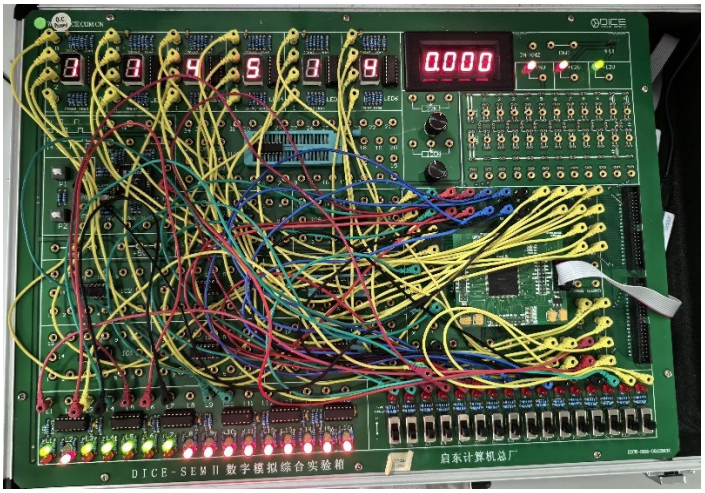


图 19 线路连接图

4.2.3 下载运行

在 Quartus II 6.0 软件的 Programmer 中选择正确的设备，点击 Start 下载成功。

4.2.4 功能验证

经过模仿仿真输入对照以及老师验收，实验箱输出与预期结果一致。

5 大型作业的心得与体会

5.1 钱鹏程

这次夏季实训的主题是常见的小型电路设备，是为了引导我们将理论知识与实际应用相结合，认识到理论在生产生活中的价值和应用方式。通过电路设计、仿真和实现验证的过程，我们提高了知识应用能力、拓展能力和自主学习能力，并促进产学研相结合。拥有分析问题、设计解决方案和实施方案的能力，才能将所学的知识真正应用于实际问题的解决和实际产品的创造。

在这次实践中，我们运用数字逻辑课程的知识，通过融会贯通的方式将其与实际问题相结合。这不仅要求我们熟练掌握基本理论知识，还需要具备创新思维和解决问题的能力。通过探索课堂外的解决方案(Verilog HDL)，我们培养了自主学习能力，并学会了根据实际需求选择和评估不同的解决方案。

在硬件开发过程中，我们利用 Quartus II 软件的强大功能，结合 Verilog 语言编程、元件设计和波形仿真等工具，进行严谨的设计、实现和验证流程。我们从简单的基础功能要求出发，逐步探索和应用更加复杂的技术。同时，为了进一步提升实际应用能力，我们将设计的在课堂要求的基础上，仿照当下数码设备多人脸识别、多指纹解锁的实际情况创新性提出了多密码的想法。而每一次的编写代码、每一次的验证、每一次的 bug 修复不仅锻炼了我们的综合能力，也给我们带来了更高的挑战和成就感。

通过这次实践，我们深刻认识到理论知识在实际生产中的重要性和实际应用方式。在解决问题的过程中，我们体会到了知识的实际价值，巩固了自己的知识水平，提高了综合能力。这次将理论知识应用于实践的经历，对我们来说是一次宝贵的学习机会，无论是硬件项目的设计流程、还是多人协作版本控制的流程，都将对我们未来的学习和职业生涯产生积极的指导作用。通过亲身实践，我们体验到了知识落地的成就感，同时也认识到了自己的潜力和发展空间。这次经历无疑将成为我们终身受益的财富。

5.2 王奕憧

团队合作的重要性：在本次项目中，我们五个人组成了一个团队，每个人都有明确的分工和职责。团队合作的过程中，我学会了如何与他人有效沟通，如何在项目遇到问题时共同讨论解决方案，以及如何在任务分配中发挥每个人的特长。通过每次的讨论和协作，我深刻认识到团队合作不仅能够提高工作效率，还能激发每个人的创新思维，使项目更好地完成。

技术应用与实践：在使用 Quartus 进行程序编写和测试的过程中，我从理论知识的学习过渡到了实际操作，这让我对数字电路设计有了更深入的理解。在项目初期，我花费了一定的时间熟悉 Quartus 软件的使用，包括如何进行逻辑设计、编写 VHDL 代码以及如何进行仿真测试。通过一次次的调试和改进，我掌握了许多实际操作的技巧，也积累了丰富的实践经验。

问题解决与自主学习：在项目过程中，我们遇到了许多意想不到的问题，比如电路设计中的逻辑错误、程序中的 bug 以及测试过程中的异常情况。每当遇到问题时，我们会共同讨论、查阅资料、寻找解决方法。这个过程中，我学会了如何在面对问题时保持冷静，通过自主学习和团队合作来解决问题。

理论与实践的结合：本次项目让我深刻体会到了理论知识在实际应用中的重要性。在课堂上学习的数字电路理论知识，在项目实践中得到了很好的应用和验证。通过将理论与实践结合，我不仅加深了对所学知识的理解，还提升了自己的动手能力和实际解决问题的能力。

总的来说，本次六位密码电子锁控制器项目让我受益匪浅。我不仅提升了自己的技术能力和实践经验，还增强了团队合作意识和项目管理能力。我相信，通过这次项目的锻炼，我将在以后的学习和工作中更加自信，也更加具备应对各种挑战的能力。

5.3 马浩元

在此次夏季大作业中，我们小组合作利用电路实现了一个六位密码锁系统。这次实验不仅复习了数字逻辑与数字逻辑实验课程中的组合电路与时序电路相关知识，还提升了我们对电路设计与实现的实际操作能力。

我们主要采用 Verilog 语言进行编码，并使用 Quartus 软件进行设计与仿真。密码锁的主要部分采用了同步时序电路设计，通过手动脉冲输入密码。整个输入过程分三次进行，每次输入两位十进制数（共八位二进制数），用于设置或输入密码。

在设计过程中，我们深刻体会到同步时序电路在电路设计中的重要作用。同步时序电路通过统一的时钟信号，使各个部件的步调一致，从而实现复杂的功能。例如，在输入密码完成后，比较模块通过时钟上升沿进行工作，将结果输出。如果密码正确，系统会显示正确的提示；如果密码错误，则会启动错误计数模块进行计数。

尽管波形仿真图在理论上是完美的，但实际操作中我们遇到了竞争与冒险问题。这些问题源于信号传递中的延迟。理论上，上升与下降沿的信号是理想的，没有误差，但在实际操作中，由于各种因素导致信号传递延迟，可能会出现仿真中未曾遇到的错误。

特别是在设计错误计数模块时，我们遇到了一个实际操作中与理论仿真不一致的问题。最初的设计是利用开启比较信号所产生的脉冲进行启动，这在波形仿真中没有问题。但在实际操作中，由于信号传递存在延迟，错误计数模块接收到脉冲时，比较结果已经变化，导致与理论预期不符，从而出现了错误。

通过此次六位密码锁的设计与实现，我深刻体会到了理论知识与实践操作之间的微妙关系，也认识到在电路设计中必须考虑各种可能的误差和延迟。理论知识是基础，但在实际操作中，我们必须考虑各种现实因素，如信号延迟、竞争与冒险等。这次实验让我认识到波形仿真虽然重要，但更关键的是如何在实际硬件环境中确保设计的稳定性和可靠性。

总的来说，这次夏季硬件大作业是一次宝贵的理论与实践结合的机会。在实践中，我们不仅发现并解决了许多问题，还复习了以往学过的知识，并解答了一些曾经存在的疑惑。这次实验让我更加明白了电路设计的严谨性和细致性，也为以后的学习打下了坚实的基础。

5.4 杨铭宇

在这次夏季实训中，我们小组成员紧密合作，共同完成了一个六位密码锁系统的设计与实现。这次实践不仅让我们复习和巩固了课堂上所学的数字逻辑知识，还极大地提升了我们的动手能力和实际操作水平。通过使用 Verilog 语言编写代码，并利用 Quartus II 软件进行仿真和验证，我们深刻体会到了理论知识与实际应用相结合的重要性。

在实训过程中，我们面对许多挑战和问题，但也因此获得了宝贵的经验。例如，在设计同步时序电路时，我们不仅要确保各个模块的正确性，还需要处理信号延迟和竞争冒险等问题。每次调试和修复 bug，都是对我们综合能力的锻炼和提升。尤其是在设计密码比较模块时，我们初次遇到的信号延迟问题，促使我们深入理解了硬件设计中的细节和复杂性。

团队合作在这次实训中发挥了关键作用。我们每个人都有明确的分工和职责，通过有效的沟通和协作，解决了许多设计和实现中的难题。讨论和合作不仅提高了工作效率，也激发了我们的创新思维，使得项目得以顺利完成。通过这次团队合作，我学会了如何在项目中发挥每个人的特长，如何有效地分配任务，以及如何在遇到问题时共同讨论解决方案。

自主学习和解决问题的能力在这次实训中也得到了极大的锻炼。我们在遇到问题时，通过查阅资料和相互讨论，寻找解决方法，不断提升了我们的知识水平和实践能力。尤其是在设计和实现多密码存储系统时，我们从实际需求出发，进行了多次调试和改进，最终成功实现了这个功能。

这次实训不仅让我巩固了理论知识，也提升了我的实际操作能力和解决问题的能力。通过实践，我更加深刻地认识到理论知识在实际应用中的重要性，并体验到了将知识应用于实际问题解决的成就感。无论是硬件设计的严谨性，还是团队协作的有效性，这次实训都为我的未来学习和职业生涯提供了宝贵的经验和指导。

总的来说，这次夏季实训是一次难得的学习机会，让我受益匪浅。通过这次项目的锻炼，我不仅提升了技术能力和实践经验，还增强了团队合作意识和项目管理能力。我相信，这次实训的经历将对我未来的学习和工作产生积极的影响，使我更加自信地应对各种挑战，继续进步。

5.5 胡铭江

为期近一月的硬件实训已接近尾声。在这次实训过程中，我同小组成员们一起学习、工作，学到了非常多有用的知识，掌握了设计简单硬件电路的方法。

我们小组所作硬件电路是六位电子密码锁的设计。在设计过程中，我们没有采用像数字逻辑实验中那样手动画电路图的方式，而是自己学习了更加高效的 verilog 语言，从最基本的门电路开始一步步向上嵌套，到基本的触发器寄存器、六位一体的寄存器组、判断模块、报警模块.....整个设计过程就如同建造大厦一般，从地基处开始，一步步累计成万丈高楼。

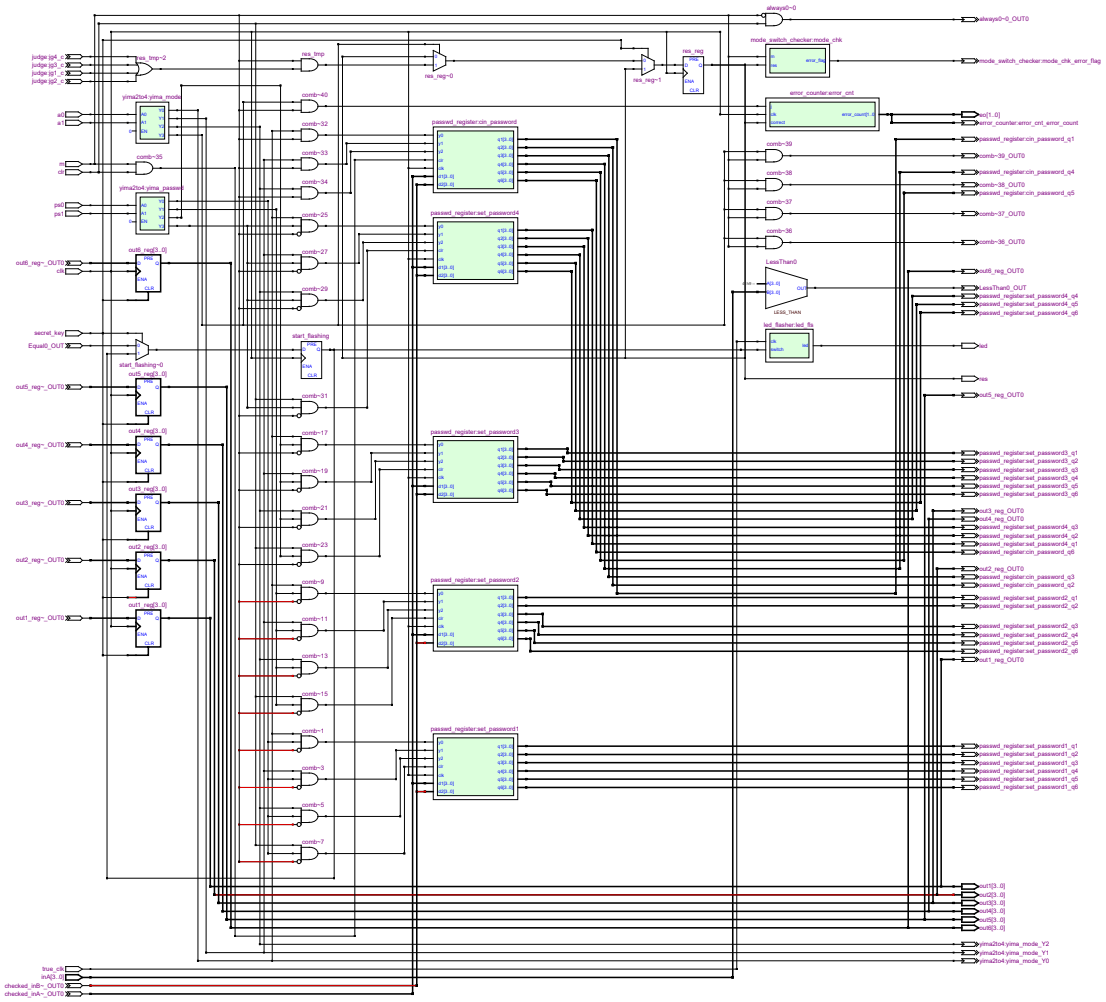
在工作中，我还提升了对 quartus 软件的使用。在自己琢磨以后才发现这个软件的强大之处，之前秋季学期在数字逻辑实验课上学到的仅仅只是九牛一毛。并且我还提升了对电路进行波形仿真的能力，了解到了更多模拟的办法，也逐渐领悟到在模拟仿真中需要怎样考虑才能尽可能涵盖更多情况的方法。

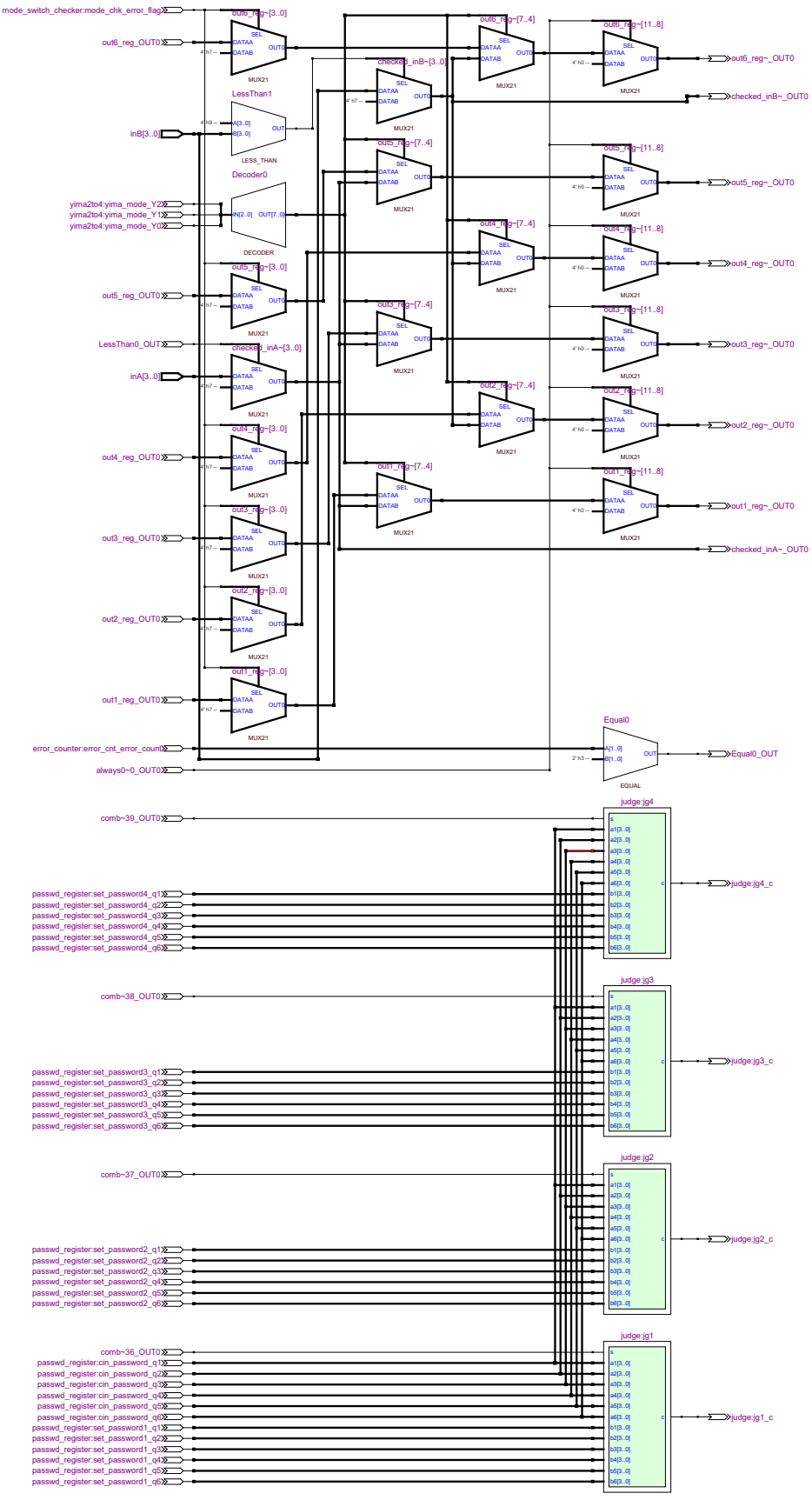
在完成实训任务中基本的要求后，我们还根据现实生活中一些经验启发，设计了一些拓展功能：比如如今的手机都能设置多个密码（或多个指纹），在此启发下我们设计了多密码存储系统；在很多登录界面输入密码是都会有隐藏密码的安全保护，因此我们也设计了密码隐私输入；此外，为了提高交互性，我们进行了多次调试，只为设计更加用户友好的效果。虽然过程并非一帆风顺，但好在我们最终还是通过努力，成功完成本次实训。

本次夏季实训的内容是跟数字逻辑密切相关的。但距离学习数字逻辑已经过去了两个学期，其中大部分内容已经有点淡忘，不过这次实训让我重新回顾了很多数字逻辑的知识。曾经在学习数字逻辑时，由于缺少实践经验，很多知识点理解的都不是很透彻，但是经过这次实训，曾经困扰我或者是似懂非懂的很多地方突然顿悟，使我切身体会到了数字电路的精妙之处。曾经我认为数字电路枯燥难懂，但是在自己亲身经历一次完整的电路设计过程之后，我发现设计数字电路其实也并非一件难事，考虑每个模块的功能，细心地进行仿真、调试，也不乏是一个有趣生动的过程。

总之，这次夏季实训令我受益匪浅。通过这次项目的锻炼我也极大提升了硬件知识储备。相信自己在未来能够更加从容面对一切挑战，继续进步。

附录




mode_switch_checker:mode_chk_error_flag
out6_reg_OUT0
inB[3..0]
yima2to4:yima_mode_Y2
yima2to4:yima_mode_Y1
yima2to4:yima_mode_Y0
out5_reg_OUT0
LessThan0_OUT
inA[3..0]
out4_reg_OUT0
out3_reg_OUT0
out2_reg_OUT0
out1_reg_OUT0
error_counter:error_cnt_error_count
always0~0_OUT0
comb~39_OUT0
comb~38_OUT0
comb~37_OUT0
comb~36_OUT0
passwd_register:cin_password_q1
passwd_register:cin_password_q2
passwd_register:cin_password_q3
passwd_register:cin_password_q4
passwd_register:cin_password_q5
passwd_register:cin_password_q6
passwd_register:set_password4_q1
passwd_register:set_password4_q2
passwd_register:set_password4_q3
passwd_register:set_password4_q4
passwd_register:set_password4_q5
passwd_register:set_password4_q6
passwd_register:set_password3_q1
passwd_register:set_password3_q2
passwd_register:set_password3_q3
passwd_register:set_password3_q4
passwd_register:set_password3_q5
passwd_register:set_password3_q6
passwd_register:set_password2_q1
passwd_register:set_password2_q2
passwd_register:set_password2_q3
passwd_register:set_password2_q4
passwd_register:set_password2_q5
passwd_register:set_password2_q6
passwd_register:set_password1_q1
passwd_register:set_password1_q2
passwd_register:set_password1_q3
passwd_register:set_password1_q4
passwd_register:set_password1_q5
passwd_register:set_password1_q6
out6_reg~_OUT0
checked_inB~_OUT0
out5_reg~_OUT0
out4_reg~_OUT0
out3_reg~_OUT0
out2_reg~_OUT0
out1_reg~_OUT0
checked_inA~_OUT0
Equal0_OUT
judge:jg4_c
judge:jg3_c
judge:jg2_c
judge:jg1_c
MUX21
DECODER
LESS_THAN
EQUAL