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: 2-7

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: 何勇乐

组员 周子俊

汪江豪

韦梁轩昂

崔丞弋

计算机工程与科学学院

报告日期 2024 年 6 月 22 日

目录

1. 实践项目意义

随着科技的发展和人们生活水平的提高，安全防护成为现代社会关注的焦点之一。电子密码锁作为一种安全、便捷的锁具，已广泛应用于家庭、办公室、酒店等场景。电子密码锁不仅可以提高安全性，还可以避免传统机械钥匙的丢失和复制问题。

在本项目中，我们小组使用 Verilog HDL 语言实现了一个电子密码锁。Verilog 是一种用于数字系统设计的语言，可以描述电子系统的结构和行为。通过 Quartus II 软件，我们将 Verilog 代码转换为对应的逻辑图，绑定对应引脚，将程序下载到数字逻辑实验箱上的 FPGA（现场可编程门阵列）芯片中，实现了电子密码锁从软件到硬件的实现。

该电子密码锁具备设置密码功能、解锁功能、输入清零功能和重置功能。通过这个项目，我们不仅初步了解了 Verilog 硬件描述语言和 Quartus II 软件的使用，还深入了解 FPGA 的工作原理和数字电路设计。该项目的意义是多方面的，包括但不限于：培养创新能力：在项目过程中，我们需要自行设计电路，实现特定功能。这要求我们充分发挥创新能力，运用所学知识解决问题。理论与实践相结合：本项目将数字电路的理论知识运用到实际硬件设计中，使我们更加深入地理解数字电路的工作原理，提高了理论联系实际的能力。增强团队协作能力：在项目实施过程中，我们需要分工合作，共同解决问题。这有助于提高我们的团队协作能力和沟通能力。提升综合素质：通过完成本项目，我们不仅掌握了专业知识，还锻炼了动手能力、解决问题能力和沟通能力，为今后的学习和工作打下了坚实基础。

在现代社会，数字电路设计和 FPGA 应用技术有着广泛的应用前景，如物联网、人工智能、自动驾驶等领域。因此，该项目对于我们未来的学习和职业发展具有重要意义。

2. 实践项目原理

2.1. 实验箱组成与原理

2.1.1. 实验箱基本组成

本次实验使用的实验箱为 DICE-SEM 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）。

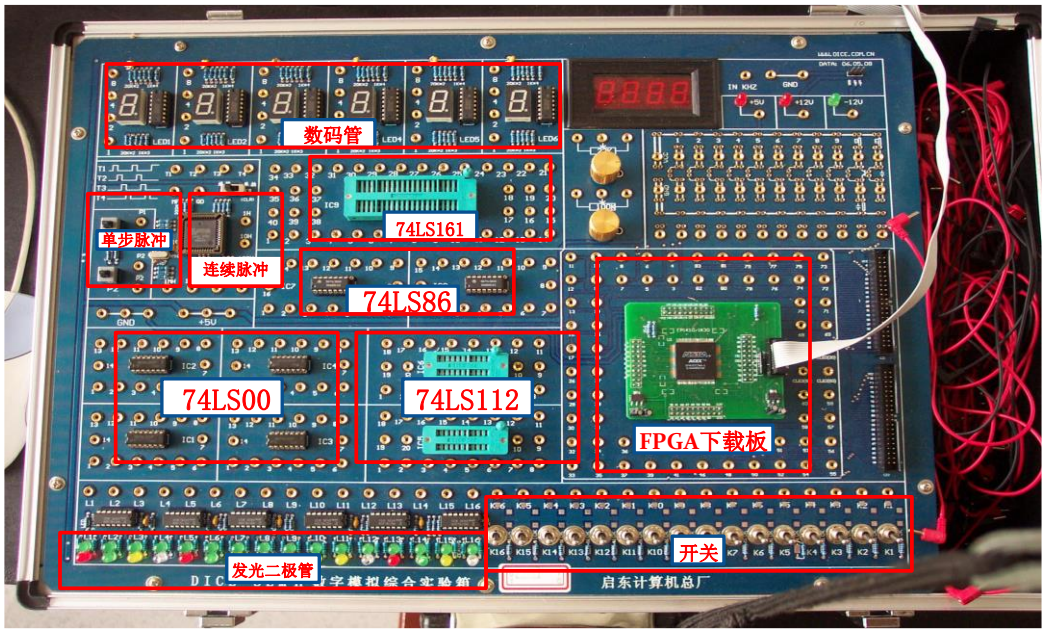


图 1 实验箱示意图

系统组成如下表：

表 1 实验箱系统组成表

电源	220V±10% ，50Hz
直流输出	±12V/200mA ；5V/2A
固定频率脉冲源 7 路	1HZ、10HZ、100HZ、1KHZ、10KHZ、100KHZ、1MHZ
数字式频率计	0~300KHZ
两组单脉冲	同时输出正负两个脉冲，脉冲幅值为 TTL 电平
独立逻辑电平开关	十六位输出“0”、“1”电平（为正逻辑）
逻辑电平显示电路	十六位由红色和绿色 LED 及驱动电路组成
BCD 码译码显示器	6 位八段 LED 数码管
Lattice-1032E	所有 I/O 口
开发式 IC 插座	40P 一个、20P 三个、16P 两个、8P 一个

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，共阳为为所有的 LED 正极接在一起。如图 2 为 1 位数码管的共阴极和共阳极原理图及其电路图：

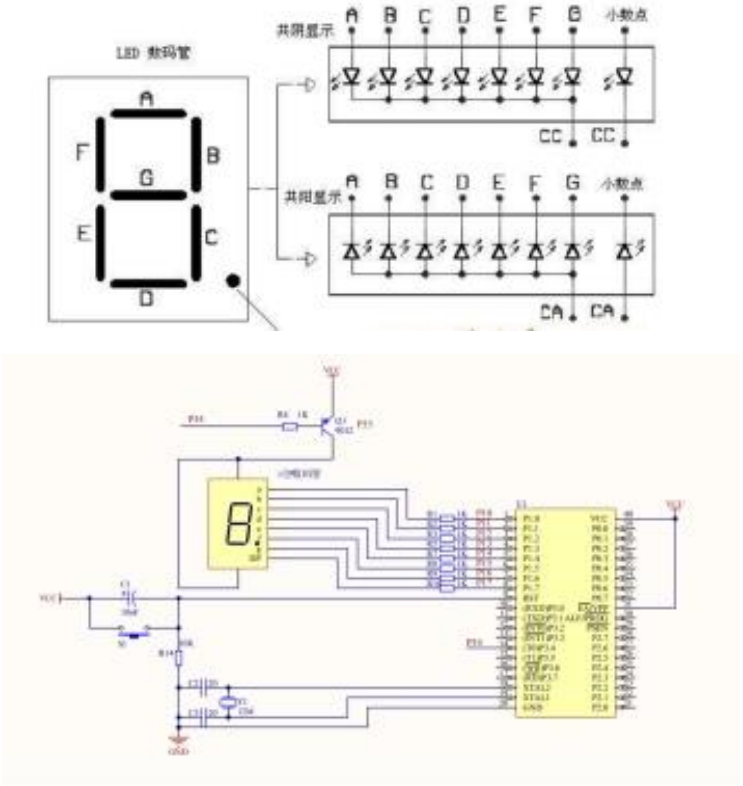
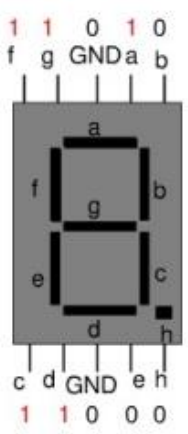


图 2 数码管原理及电路图

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，下图以显示数字‘5’为例，编码为 0x6D。如果数码管为共阳极，只需要对共阴极的编码做一个取反操作即可。



共阴极数码管显示字段

显示字符	码 g	f	e	d	c	b	a	十六进制
0	0	1	1	1	1	1	1	3F
1	0	0	0	0	1	1	0	06
2	1	0	1	1	0	1	1	5B
3	1	0	0	1	1	1	1	4F
4	1	1	0	0	1	1	0	66
5	1	1	0	1	1	0	1	6D
6	1	1	1	1	1	0	1	7D
7	0	0	0	0	1	1	1	07
8	1	1	1	1	1	1	1	7F
9	1	1	0	1	1	1	1	6F
A	1	1	1	0	1	1	1	77
b	1	1	1	1	1	0	0	7C
c	0	1	1	1	0	0	1	39
d	1	0	1	1	1	1	0	5E
E	1	1	1	1	0	0	1	79
F	1	1	1	0	0	0	1	71

图 3 八段 LED 数码管显示字段

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：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 七段数码管译码器

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。

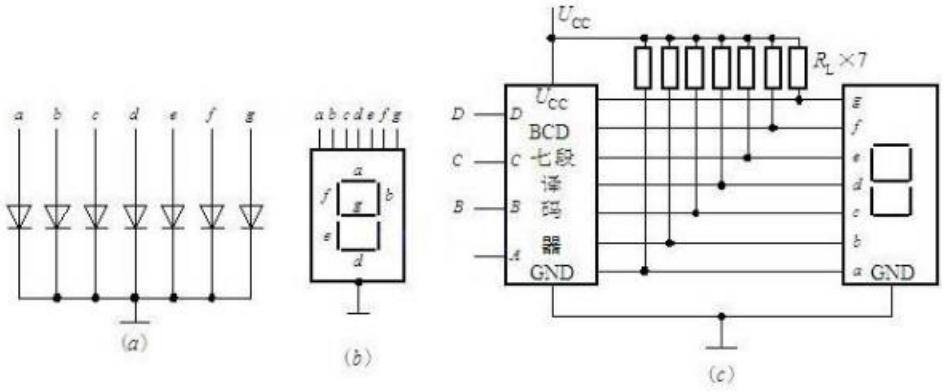


图 4 BCD 七段数码管译码器

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)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 ， 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 Fa~Fg=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 BCD 七段数码管真值表

以下是实现 8421BCD 七段译码功能的电路图

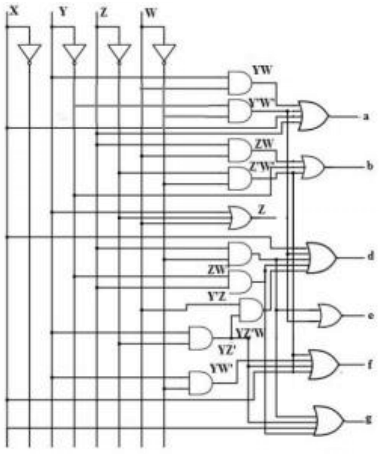


图 6 BCD 七段数码管译码功能电路

2.2. Verilog HDL

2.2.1. Verilog 模块

模块（module）是 verilog 最基本的概念，是 v 设计中的基本单元，每个 v 设计的系统中都由若干 module 组成。模块在语言形式上是以关键词 module 开始，以关键词 endmodule 结束的一段程序。模块的实际意义是代表硬件电路上的逻辑实体。每个模块都实现特定的功能。模块的描述方式有行为建模和结构建模之分。模块之间是并行运行的。模块是分层的，高层模块通过调用、连接低层模块的实例来实现复杂的功能。各模块连接完成整个系统需要一个顶层模块（top-module）。

2.2.2. 数据类型 wire、reg

wire 的本质是一条没有逻辑的连线。wire 型数据常用来表示以 assign 关键字指定的组合逻辑信号，模块的输入输出端口类型都默认为 wire 型，wire 相当于物理连线，默认初始值是 z（高组态）。

reg 型表示的寄存器类型，用于 always 模块内被赋值的信号，必须定义为 reg 型，代表触发器，常用于时序逻辑电路，reg 相当于存储单元，默认初始值是 x（未知状态）。reg 型相对复杂些，其综合后的输出主要还看具体使用的场景：在组合电路中使用 reg，合成后的仍然是 net 网络；在时序电路中使用 reg 合成后的才是 register。

2.2.3. 关键字 always

always 块是 Verilog 中用来描述组合逻辑以及时序逻辑的语法。always 块内的敏感列表控制内部语句的触发时机。可以理解为一种定时，如果没有了敏感列表，则为零延迟，那么就会不断地触发。

3. 实践项目设计

3.1. 元件各模块设计

3.1.1. 边沿检测设计

该状态机以固定时钟频率运行，在每个时钟周期检查各个控制端是否处于边沿状态决定状态的改变，额外编写了一个带缓冲的边沿检测模块。设置两个寄存器，对前一状态和后一状态进行寄存，若前后两个状态不同，则检测到了边沿。对于上升沿和下降沿的确定可以用组合逻辑比较来确定。若前一状态 D[1]为高电平，后一状态 D[0]为低电平，则为下降沿，反之为上升沿，代码如下。

```
module Watcher (
    input  wire clk,
    input  wire rst_n,
    input  wire signal,
    output wire pos,
    output wire neg
);

  reg [2:0] state;

  always @(posedge clk, negedge rst_n) begin
    if (!rst_n) begin
      state[0] <= 1'b0;
      state[1] <= 1'b0;
      state[2] <= 1'b0;
    end else begin
      state[0] <= signal;
      state[1] <= state[0];
      state[2] <= state[1];
    end
  end

  assign pos = state[1] & !state[2];
  assign neg = !state[1] & state[2];
endmodule
```

每个需要触发状态变化的输入信号都需要进行边沿检测，在 Lock 锁主模块中使用 Watcher 模块对各种输入带来的变化进行边沿检测，代码如下。

```
  // 边沿检测
/* verilator lint_off UNUSEDSIGNAL */
wire modeSettingWirePos, modeSettingWireNeg;
Watcher modeSettingWireWatcher (
    .clk   (clk),
    .rst_n (rst_n),
    .signal(controlSignal[3]),
    .pos   (modeSettingWirePos),
    .neg   (modeSettingWireNeg)
);
wire inputNextSignalPos, inputNextSignalNeg;
Watcher inputNextSignalWatcher (
    .clk   (clk),
    .rst_n (rst_n),
    .signal(controlSignal[2]),
    .pos   (inputNextSignalPos),
    .neg   (inputNextSignalNeg)
);
wire resetSignalPos, resetSignalNeg;
Watcher resetSignalWatcher (
    .clk   (clk),
    .rst_n (rst_n),
    .signal (controlSignal[1]),
    .pos   (resetSignalPos),
    .neg   (resetSignalNeg)
);
wire passwordInputFinishSignalPos, passwordInputFinishSignalNeg;
Watcher passwordInputFinishSignalWatcher (
    .clk   (clk),
    .rst_n (rst_n),
    .signal(controlSignal[0]),
    .pos   (passwordInputFinishSignalPos),
    .neg   (passwordInputFinishSignalNeg)
);
/* verilator lint_off UNUSEDSIGNAL */
```

3.1.2. 密码输入设计

根据需求和实验箱开关数量的限制，难以一次将六位密码全部输入。通过预定义常量控制，输入以每四位二进制一组、一次输入一到多组（可控）的形式输入密码。每输入一次，输入指针向后移动一段，直到循环。代码如下。

```
StateInput: begin
```

```
   begin
     inputBuffer = ((inputBuffer | maskForEdit) ^ maskForEdit) |
patchedNumber;
     inputBufferIndex = inputBufferIndex == 0 ? InputBufferSlots -
1 : inputBufferIndex - 1;
   end
   if (stateStack == StateSet) begin
      state = StateSet;
    end else if (stateStack == StateUnlock) begin
      state = StateUnlock;
    end
         stateStack = StateNIL;
 end
```

3.1.3. 输入缓存模式设计

设计输入缓存模式，输入内容先进入缓存寄存器，在切换模式时，才会将缓存写回密码储存或是用于检验。不难看出，验证和输入模式下都需要输入密码和清空密码的功能，因此进入输入密码或清空密码的状态前，通过一个栈寄存器保存之前的状态，完成输入或清空后回到栈寄存器中保存的状态，实现了状态的复用。

3.1.4. 遮罩设计

为实现输入位闪烁的功能，通过一个遮罩将正在输入的数位与 0 之间闪烁，同时为了使得下一位输入仍保持上次输入的结果，当前遮罩中的显示用缓存寄存器将保留上次的输入缓存内容，代码如下。

```
// 计算输入偏移量与输入遮罩
begin
  shift = inputBufferIndex * `INPUT_BITS;
  begin
    patchedNumber                    = 0;
    patchedNumber[shift+:`INPUT_BITS] = inputNumber;
  end
end
maskForEdit = InputMask << shift;

//闪烁显示输入遮罩
if (!onDisplayMessage) begin
  displayBuffer = inputBuffer;
```

```
  // 闪烁显示当前输入的位
  if (!onDisplayMessage) begin
    onEditFlashCounter = onEditFlashCounter+1 ==
2*`CLOCKS_PER_FLASH?0:onEditFlashCounter+1;

    displayBuffer = displayBuffer & (~maskForEdit);
    if (onEditFlashCounter < `CLOCKS_PER_FLASH) begin
      displayBuffer = displayBuffer | patchedNumber;
    end
  end
  // 只显示最后一次输入的几位
  if (state == StateUnlock || stateStack == StateUnlock) begin
    displayBuffer = displayBuffer & (maskForEdit);
  end
end
```

3.1.5. 解锁正确或错误的闪烁

验证模式下，验证密码的上升沿到达时，进入验证状态，并决定之后进入密码正确提示或密码错误提示状态。并设置标志变量，以告知显示部分正在显示提示内容。

进入提示状态时，会开启一组递增的寄存器，记录提示状态下经过的周期数，达到一定周期后切换提示内容，实现闪烁。密码错误提示状态会额外记录进入状态的次数，达到三次时显示不同的错误提示，代码如下。

```
StateUnlockIncorrect: begin
  ErrorTimes = ErrorTimes + 1;
  if (displayMessageClockCounter % (2 * `CLOCKS_PER_FLASH) == 0)
begin
    lightErrorOneTimesBuffer = 1;
    if (ErrorTimes == 2'b11) begin
      displayBuffer = `MESSAGE_INCORRECT;
    end
  end else if (displayMessageClockCounter % `CLOCKS_PER_FLASH == 0)
begin
    lightErrorOneTimesBuffer = 0;
    if (ErrorTimes == 2'b11) begin
      displayBuffer = 0;
    end
  end

  if (displayMessageClockCounter == 0) begin
```

```
    onDisplayMessage = 0;
    state           = StateUnlock;
  end else begin
    displayMessageClockCounter = displayMessageClockCounter - 1;
  end
end
```

3.2. FSM 设计

3.2.6. FSM 有限状态机

有限状态机（Finite-state machine，FSM），又称有限状态自动机，简称状态机，是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。FSM 是一种算法思想，简单而言，有限状态机由一组状态、一个初始状态、输入和根据输入及现有状态转换为下一个状态的转换函数组成。其作用主要是描述对象在它的生命周期内所经历的状态序列，以及如何响应来自外界的各种事件。

我们绘制了一个当前项目的 FSM 状态转移图示，如下图。我们一共设计了七种状态，分别是 StateSet 对应密码设置模式、StateInput 对应密码输入、StateUnlock 对应密码解锁模式、StateReset 对应重置当前输入、StateUnlockInputFinish 对应解锁模式下输入结束、StateUnlockCorrect 对应解锁模式下密码输入正确、StateUnlockInCorrect 对应解锁模式下密码输入错误。

图 7 FSM 状态转移图

3.2.7. FSM 转移情况

密码锁开机默认进入状态 StateSet。

①StateSet：在该模式下可以沿触发 inputNextSignal 转移到 StateInput；调节 modeSettingWire 的值转移到 StateUnlock；上升沿触发 resetSignal 转移到 StateReset。

②StateInput：该模式不存在转出的情况，只要进入到了设置密码模式以及解锁密码模式并沿触发 inputNextSignal 就会转移到 StateInput 状态，这个状态更多是被包含在 StateSet 和 StateUnlock 当中的。

③StateUnlock：该模式下设置 modeSettingWire 的值可以转移到 StateSet；升沿触发 resetSignal 转移到 StateReset；上升沿触发 passwordInputFinishSingal 转移到 StateUnlockInputFinish 状态。

④StateReset：该状态触发后将会根据当前状态栈顶进行返回，可以返回到 StateUnlock 或 StateSet 状态。

⑤ StateUnlockInputFinish ： 该 状 态 是 在 StateUnlock 触 发 passwordInputFinishSignal 后转移进来，用于中转判断当前密码解锁是否正确，当 inputBuffer 等于设置好的密码时，转移到 StateUnlockCorrect 状态，否则转移到 StateUnlockIncorrect 状态。

⑥StateUnlockCorrect：该状态表示解锁模式下密码输入正确，此时进行正确的闪烁，由 displayMessageClockRest 的值进行控制，当该值变为 0，也即停止闪烁时，转移回解锁模式 StateUnlock。

⑦StateUnlockIncorrect：该状态表示解锁模式下密码输入错误，此时进行错误的闪烁报警，由 displayMessageClockRest 的值进行控制，当该值变为 0，也即停止闪烁时，转移回解锁模式 StateUnlock。在该情况下，还有额外的一个计数器记录错误次数，当错误超过三次时数码管进行特殊错误闪烁报警。

3.3. Testbench 设计

3.3.8. Lock 模块实例化

我们需要对设计好的 Lock 模块进行实例化，然后在 testbench 文件中对该实例进行各种不同输入模拟进行情景模拟，进行仿真。在实例化 Lock 模块前，也需要定义各种 IO 接口对应于 Lock 的输入和输出。我们将各种输入输出信号加上后缀“_tb”进行统一命名，表示这是 testbench 文件下的。

在这里，我们定义了 clk_tb 作为时序、rst_n_tb 作为状态重置、modeSettingWire_tb 作为密码锁模式设置、inputNextSignal_tb 作为循环输入下两位的沿触发信号、resetSignal_tb 作为重置当前输入的上升沿触发信号、passwordInputFinishSignal_tb 作为当前输入结束的上升沿触发信号、inputNumber_tb 作为当前正在输入的两个数码管的数值、display_tb 作为所有数码管的显示值、lightErrorOneTimes_tb 作为每次错误输入的显示值。代码如下

```
reg clk_tb;
reg rst_n_tb;
reg modeSettingWire_tb;
reg inputNextSignal_tb;
reg resetSignal_tb;
reg passwordInputFinishSignal_tb;
`INPUT_TYPE inputNumber_tb;

/* verilator lint_off UNUSEDSIGNAL */
wire [`PASSWORD_BITS-1:0] display_tb;
```

```
  wire lightErrorOneTimes_tb;
  /* verilator lint_off UNUSEDSIGNAL */

  Lock lock (
      .clk(clk_tb),
      .rst_n(rst_n_tb),
      .controlSignal({
        modeSettingWire_tb, inputNextSignal_tb, resetSignal_tb,
passwordInputFinishSignal_tb
      }),
      .inputNumber(inputNumber_tb),
      .display(display_tb),
      .lightErrorOneTimes(lightErrorOneTimes_tb)
  );
```

3.3.9. 设置时钟

由于需要进行各种闪烁显示，我们需要设定时钟，这里我们使用了 1e6 ns 的时钟频率，并且在每次操作之间添加一些延迟，这些延迟是 1e9 ns。时钟则每间隔 $5*10^5$ ns 进行跳变，代码如下。

```
  localparam integer NsPerClk = 1000000;
  localparam integer NsPerOp = 1000 * NsPerClk;
  localparam integer InputSignalDelay = 100 * NsPerClk;
  always #(NsPerClk / 2) clk_tb = ~clk_tb;
```

3.3.10. 各种输入的初始化

一开始，我们并没有正确地对输入进行初始化，导致仿真时出现了很多不确定值的情况，即仿真显示为 X，所以我们将几乎所有能初始化的值先进行初始化再进行不同输入变化模拟实际使用情景，代码如下。

```
    // 随机密码
    randomPassword = 24'hCCAA11;

    // 初始化
    fork
      clk_tb = 0;
      modeSettingWire_tb = 1'b0;
      inputNextSignal_tb = 1'b0;
      inputNumber_tb = 8'h00;
      resetSignal_tb = 1'b0;
```

```
    passwordInputFinishSignal_tb = 1'b0;
    rst_n_tb = 1'b0;
  join
  #NsPerOp;
```

3.3.11. 开始执行不同输入变化

将输入发生变化模拟各种使用情景，例如我们将 modeSettingSignal_tb 先设置为 0 进入设置密码模式，然后改变 inputNumber_tb 的值，将 inputNextNumberSignal_tb 设置沿触发，模拟循环输入密码。之后我们将 resetSignal_tb 设置为上升沿模拟重置当前输入，再将之前随机产生的密码进行输入，设置好密码后，则将 modeSettingSignal_tb 设置为 1 进入解锁密码模式，并尝试正确解锁密码、错误解锁密码、三次错误解锁密码等情景，这里只有部分参考代码。

```
  // 开始执行
  fork
    rst_n_tb = 1'b1;
  join
  #NsPerOp;

  // 设置
  // 输入密码
  fork
    inputNumber_tb     = 'hCC;
    inputNextSignal_tb = #InputSignalDelay 1'b1;
  join
  #NsPerOp;

  fork
    inputNumber_tb     = 'hAA;
    inputNextSignal_tb = #InputSignalDelay 1'b0;
  join
  #NsPerOp;

  fork
    inputNumber_tb     = 'h11;
    inputNextSignal_tb = #InputSignalDelay 1'b1;
  join
  #NsPerOp;

  fork
```

```
    inputNumber_tb     = 'h22;
    inputNextSignal_tb = #InputSignalDelay 1'b0;
  join
  #NsPerOp;

  // 重置密码
  fork
    resetSignal_tb = 1'b1;
  join
  #NsPerOp;
```

4. 仿真模拟

4.1. 根据时钟频率当前输入位闪烁

4.1.1. 情景——设置密码模式

在设置密码模式下，我们需要将 controlSignal 从高位到低位对应 modeSettingWire_tb,inputNextSignal_tb,resetSignal_tb,passwordInputFinishSignal_tb，此时全部置 0，Reset 置为 1，passwordInputFinishSignal 置 0，所有位不发生沿触发。如下表 1 所示。此时将 inputNumber 设置为十六进制 $(CC)_h$。

表 2 设置密码模式各输入

variable	controlSignal[3..0]	rst_n	passwordInputFinishSignal	inputNumber
initial	0 0 0 0	1	0	$(CC)_h$

在仿真模拟下，可以看见当前输入位从 CC 到 00 间不断循环。如图。

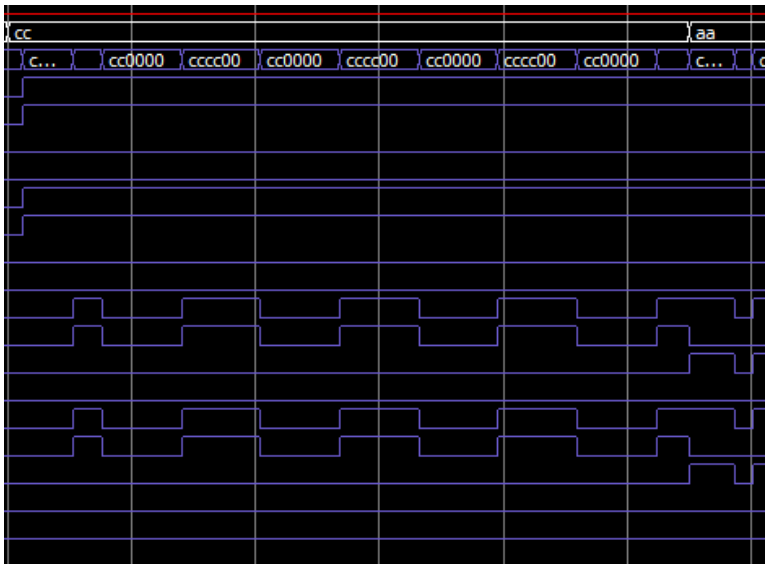


图 8 输入位闪烁

4.1.2. 情景——解锁密码模式

在解锁密码模式下，我们需要将 controlSignal 从高位到低位对应 modeSettingWire_tb,inputNextSignal_tb,resetSignal_tb,passwordInputFinishSignal_tb。此时除 modeSettingWire_tb 置 1 外全部置 0，Reset 置为 1，passwordInputFinishSignal 置 0，所有位不发生沿触发。如下表 3 所示。此时将 inputNumber 设置为十六进制 $(11)_h$。

表 3 解锁密码模式各输入

variable	controlSignal[3..0]	rst_n	passwordInputFinishSignal	inputNumber
initial	1 0 0 0	1	0	$(11)_h$

在仿真模拟下，可以看见当前输入位从 11 到 00 间不断循环。如图 。

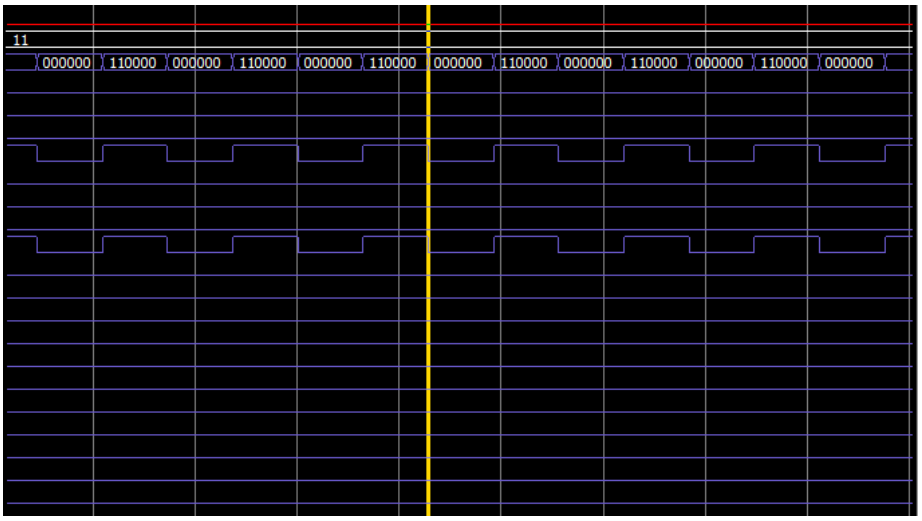


图 9 解锁模式输入位闪烁

4.2. InputNextSignal 信号沿触发循环移位输入

4.2.1. 情景——设置密码模式

在设置密码模式下，我们需要将 controlSignal 设置为设置密码模式，其余位置 0，Reset 置为 1，passwordInputFinishSignal 置 0，所有位不发生沿触发。如下表 4 所示。此时将 inputNumber 设置为十六进制 $(aa)_h$。同时还需要让 InputNextSignal 发生沿变化。

表 4 设置密码模式输入下一位

variable	Initial or change
controlSignal[3..0]	0 0 0 0
rst_n	1
passwordInputFinishSignal	0
inputNumber	$(aa)_h$
InputNextSignal	↓

在仿真模拟下，可以看见当前输入位从 aa 到 00 间不断循环。如图 蓝色曲线代表 InputNextSignal 从 1 到 0 发生下降沿触发，而白色曲线代表 InputNumber 稳定在 $(aa)_h$ 。可以看见，在沿触发发生前，当前输入位是第 3、4 位，此时第 3、4 位闪烁；而沿触发发生后，$(aa)_h$ 被输入到第 3、4 位，继续闪烁的是第 5、6 位。

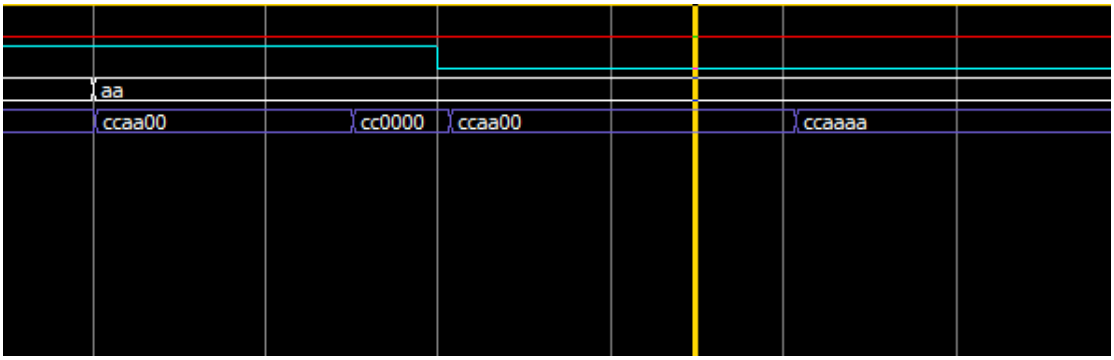


图 10 设置密码模式输入下一位

4.2.2. 情景——解锁密码模式

在解锁密码模式下，我们需要将 controlSignal 设置为解锁密码模式，其余位置 0，Reset 置为 1，passwordInputFinishSignal 置 0，所有位不发生沿触发。如下表 5 所示。此时将 inputNumber 设置为十六进制 $(cc)_h$。同时还需要让 InputNextSignal 发生沿变化。

表 5 解锁密码模式输入下一位

variable	Initial or change
controlSignal[3..0]	1 0 0 0
rst_n	1
passwordInputFinishSignal	0
inputNumber	$(cc)_h$
InputNextSignal	↓

在仿真模拟下，可以看见当前输入位从 cc 到 00 间不断循环。如图蓝色曲线代表 InputNextSignal 从 1 到 0 发生下降沿触发，而白色曲线代表 InputNumber 稳定在 $(cc)_h$ 。可以看见，在沿触发发生前，当前输入位是第 1、2 位，此时第 1、2 位闪烁；而沿触发发生后，$(cc)_h$ 被输入到第 1、2 位，继续闪烁的是第 3、4 位。在输入完成后，第 1、2 位常亮为 0，而第 3、4 位则在上次的输入 $(cc)_h$ 与 00 间闪烁。

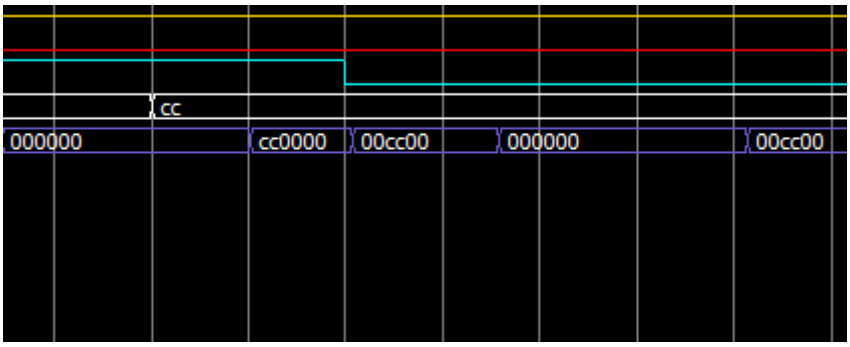


图 11 解锁模式输入下一位

4.3. ResetSignal 信号上升沿触发重置当前输入情况

4.3.1. 情景——设置密码模式

在设置密码模式下，我们需要将 controlSignal 设置为设置密码模式，其余位置 0，Reset 置为 1，passwordInputFinishSignal 置 0，所有位不发生沿触发。如下表 6 所示。此时将 inputNumber 设置为十六进制 $(22)_h$。同时 resetSignal 发生上升沿变化。

表 6 设置模式触发 ResetSignal

variable	Initial or change
controlSignal[3..0]	0 0 0 0
rst_n	1
passwordInputFinishSignal	0
inputNumber	$(22)_h$
InputNextSignal	0
resetSignal	↑

在仿真模拟下，可以看见当前输入位从 22 到 00 间不断循环。如图蓝色曲线代表 InputNextSignal 稳定为 0，紫色曲线代表 resetSignal 从 0 变成 1 的上升沿，而白色曲线代表 InputNumber 稳定在 $(22)_h$ 。可以看见，在上升沿发生前，当前输入位是第 3、4 位，此时第 3、4 位闪烁；而上升沿触发后，当前输入位变为最高两位，即 1、2 位，此时 1、2 位开始闪烁，而其他位全部为 0。

图 12 设置模式触发 ResetSignal

4.3.2. 情景——解锁密码模式

在解锁密码模式下，我们需要将 controlSignal 设置为解锁密码模式，其余位置 0，Reset 置为 1，passwordInputFinishSignal 置 0，所有位不发生沿触发。如下表 7 所示。此时将 inputNumber 设置为十六进制 $(11)_h$。同时 resetSignal 发生上升沿变化。

表 7 解锁模式触发 ResetSignal

variable	Initial or change
controlSignal[3..0]	1 0 0 0
rst_n	1
passwordInputFinishSignal	0
inputNumber	$(11)_h$
InputNextSignal	0
resetSignal	↑

在仿真模拟下，可以看见当前输入位从 11 到 00 间不断循环。蓝色曲线代表 InputNextSignal 稳定为 0，紫色曲线代表 resetSignal 从 0 变成 1 的上升沿，而白色曲线代表 InputNumber 稳定在 $(11)_h$ 。可以看见，在上升沿发生前，当前输入位是第 3、4 位，此时第 3、4 位闪烁；而上升沿触发后，当前输入位变为最高两位，即 1、2 位，此时 1、2 位开始闪烁，而其他位全部为 0。

4.4. 解锁密码错误

4.4.1. 情景——单次解锁错误

我们需要将 controlSignal 设置为解锁密码模式，其余位置 0，Reset 置为 1，passwordInputFinishSignal 置 0，同时，我们需要先在设置模式下将密码设置为 $(ccaa11)_h$。如下表 8 所示。然后我们输入一串与$(ccaa11)_h$ 相异的密码，再将 passwordInputFinishSignal 发生上升沿变化。

表 8 单次解锁错误

Variable	Initial or change
controlSignal[3..0]	1 0 0 0
rst_n	1
passwordInputFinishSignal	↑
inputNumber	$(002233)_h$

在仿真模拟下，可以看见当前输入位从 33 到 00 间不断循环。红色曲线代表单次错误时的交变信号，可以看见，在上升沿发生前，当前输入位是第 1、2 位，此时第 1、2 位闪烁；而上升沿触发后，当前 display 全部停止闪烁，并且全显式为刚才的输入值，同时由红色曲线的 lightErrorOneTimes 信号闪烁。如图

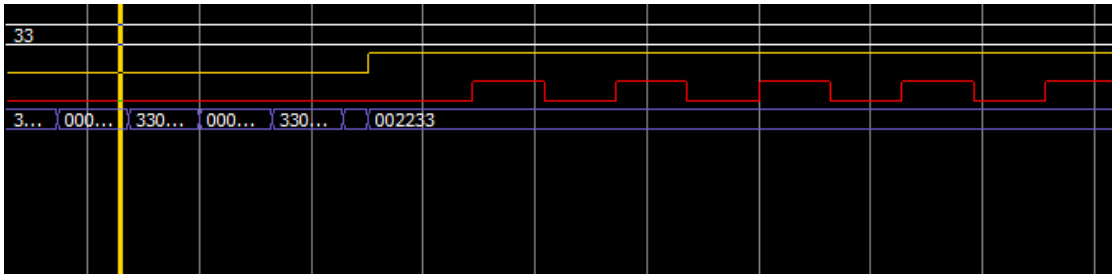


图 13 单次解锁错误

4.4.2. 情景——连续三次解锁错误

将 controlSignal 设置为解锁密码模式，其余位置 0，Reset 置为 1，passwordInputFinishSignal 置 0，同时，我们需要先在设置模式下将密码设置为$(ccaa11)_h$。如下表 9 所示。然后我们输入一串与$(ccaa11)_h$相异的密码，并重复三次 passwordInputFinishSignal 发生上升沿变化。

表 9 连续三次解锁错误

Variable	Initial or change
controlSignal[3..0]	1 0 0 0
rst_n	1
passwordInputFinishSignal	↑↑↑
inputNumber	$(330022)_h$

在仿真模拟下，可以看见当前输入位从 33 到 00 间不断循环。红色曲线代表单次错误时的交变信号，可以看见，在上升沿发生前，当前输入位是第 3、4 位，此时第 3、4 位闪烁；而上升沿触发后，当前 display 全部停止闪烁，并且全

显式为刚才的输入值，同时由红色曲线的 lightErrorOneTimes 信号闪烁，在一定时间之后，display 全部开始从$(000000)_h$到$(111111)_h$闪烁。如图。

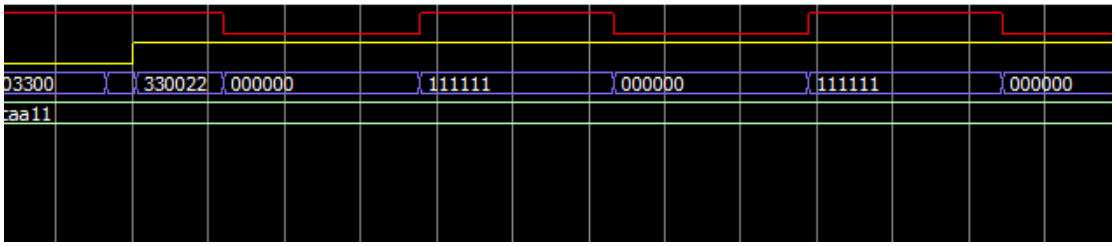


图 14 连续三次解锁错误闪烁

该闪烁时间与 lightErrorOneTimes 的闪烁时间一致。当 lightErrorOneTimes 停止闪烁时，display 也全部停止错误闪烁，并转入原来的解锁状态。如图。

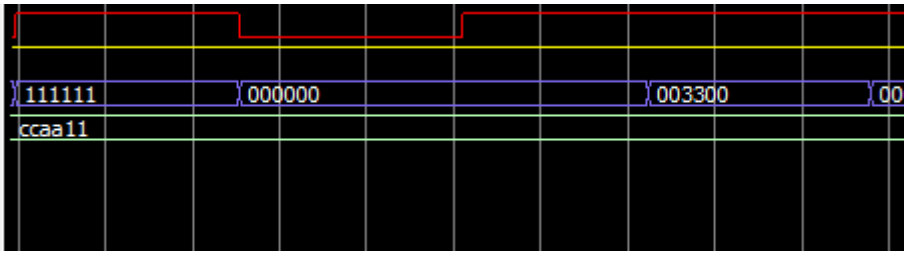


图 15 连续三次解锁错误停止闪烁

5. 实践项目调试过程

5.1. 线路连接

在 Quartus II 创建好的项目中编译当前 Verilog 文件，并为当前 Verilog 文件创建 Symbol，之后打开图形编辑器，导入创建的 Symbol，并连接引脚。如图。

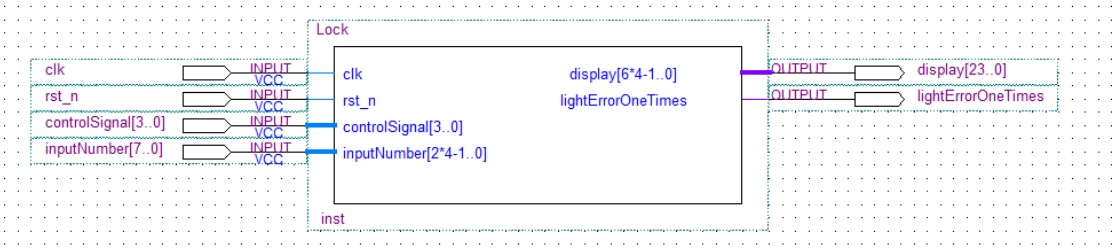


图 16 线路连接

5.2. 引脚设置

打开 Assignments – Device 选择设备，根据实验箱的 FPGA 芯片型号，选择 EP1K30TC144-3，之后在 Assignments – Pins 选择设置引脚。结果如下图

图 17 引脚设置

实验箱 ISP1032 引脚与 Quartus II EP1K30 引脚对应关系有如下表格。

表 10 ISP1032 与 Quartus II EP1K30 引脚对应关系

实验箱 ISP1032 引脚	Quartus II 软件 EP1K10、EP1K30 引脚	实验箱 ISP1032 引脚	Quartus II 软件 EP1K10、EP1K30 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

根据上述表格对输入端口进行连线，如下表。

表 11 输入端引脚设置

端口	芯片引脚	实验箱引脚	所连接开关
clk	PIN_132	PIN_6	100Hz
controlSignal[0]	PIN_9	PIN_12	K1
controlSignal[1]	PIN_119	PIN_80	K2
controlSignal[2]	PIN_120	PIN_83	K3
controlSignal[3]	PIN_13	PIN_13	K4
inputNumber[0]	PIN_49	PIN_45	K6
inputNumber[1]	PIN_51	PIN_46	K7
inputNumber[2]	PIN_59	PIN_47	K8
inputNumber[3]	PIN_60	PIN_48	K9
inputNumber[4]	PIN_62	PIN_49	K10
inputNumber[5]	PIN_63	PIN_50	K11
inputNumber[6]	PIN_64	PIN_51	K12
inputNumber[7]	PIN_65	PIN_52	K13
rst_n	PIN_121	PIN_82	K15

对输出端口进行连线，如下表。

表 12 输出端引脚设置

端口	芯片引脚	实验箱引脚	所连接数码管
display[0]	PIN_18	PIN_15	Light_0
display[1]	PIN_19	PIN_16	Light_0
display[2]	PIN_131	PIN_3	Light_0
display[3]	PIN_21	PIN_17	Light_0

display[4]	PIN_130	PIN_4	Light_1
display[5]	PIN_23	PIN_18	Light_1
display[6]	PIN_26	PIN_26	Light_1
display[7]	PIN_27	PIN_27	Light_1
display[8]	PIN_117	PIN_78	Light_2
display[9]	PIN_29	PIN_28	Light_2
display[10]	PIN_30	PIN_29	Light_2
display[11]	PIN_118	PIN_81	Light_2
display[12]	PIN_32	PIN_30	Light_3
display[13]	PIN_33	PIN_31	Light_3
display[14]	PIN_36	PIN_32	Light_3
display[15]	PIN_37	PIN_33	Light_3
display[16]	PIN_38	PIN_34	Light_4
display[17]	PIN_39	PIN_35	Light_4
display[18]	PIN_41	PIN_36	Light_4
display[19]	PIN_43	PIN_37	Light_4
display[20]	PIN_44	PIN_38	Light_5
display[21]	PIN_46	PIN_39	Light_5
display[22]	PIN_47	PIN_40	Light_5
display[23]	PIN_48	PIN_41	Light_5

5.3. 线路连接

完成引脚设置后，根据上述设置将线路进行连接，连接后如图所示。

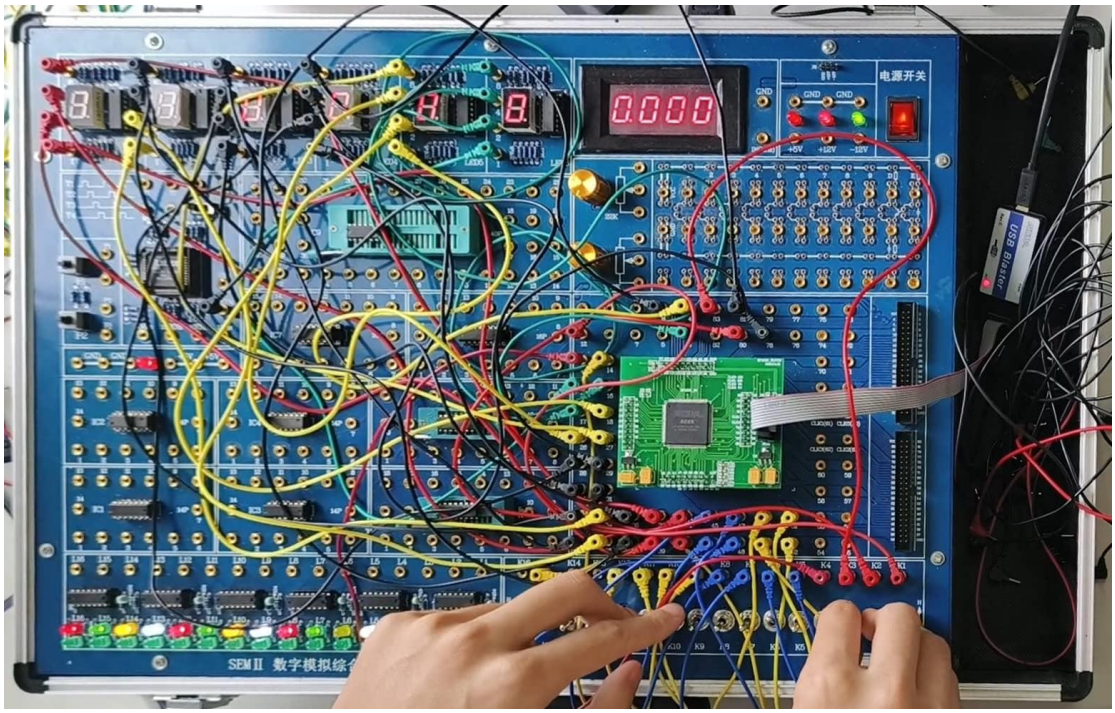


图 18 连接线路图

5.4. 下载运行

完成线路连接后，打卡 Quartus II Tools – Programmer ，勾选 Program/Configure 并检查是否存在硬件，如未检查到，需要重新尝试连接 FPGA 芯片。如可以检查到硬件，则点击 Start，下载至 FPGA 芯片中。如图。

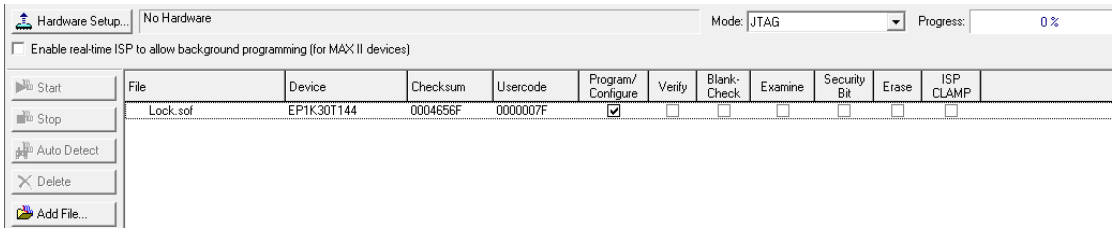


图 19 下载至 FPGA 芯片中

6. 大型作业的心得与收获

6.1. 崔丞弋

在本次的计算机硬件综合大型作业中，我们小组通过合作成功完成了刘威密码电子锁控制器，让我收获了很多。

在本次大作业中，我自秋季学期结束以后又一次使用了 DICE-SEM 数字模拟综合实验箱，所以对实验箱硬件的了解变得更为深刻。本次作业的最终展示形式是通过实验箱上进行，所以对于实验箱的组成，有了充分的机会进行复习与回顾，对于各个部分各自的作用也可以有更明确的认识。当然这些认识以及了解通过本次实训中也进行了应用。另外，这次大作业主要涉及 FPGA 芯片，该芯片具有可编程性，所以也让我对于 Quartus II这个软件的使用有了更多的了解。并且，该芯片广泛用于嵌入式系统、通信、工业控制、科学研究等领域，所以本次实训也让我对于如何在其他硬件中 FPGA 的作用有了更为充分的了解。

本次大作业中我们使用了 Verilog 语言来进行编程。让我对于如何使用、如何理解 Verilog 语言两方面都有所收获。从使用方面来说，通过本次大作业，我第一次对 Verilog 语言的基础语法进行了了解，了解了模块和逻辑块的使用方法；从理解方面来说，本次作业需要将 Verilog 编程应用在 Quartus II软件中，由于我们是将其他组员编写好的代码放入 Quartus II进行了运行，所以我能够在正确运行前对代码中使用的变量类型有更为充分的认识，比如为什么有的变量只能定义成 wire 型而不能定义成 reg 型，Quartus II并不支持 fork 和 join，所以需要检查代码块之间不能出现并行执行的需求等。

而在编译成功后，我们还对于代码进行了仿真模拟，以检查代码内的逻辑是否还存在问题，所以通过本次大作业，我还了解了 ModuleSim 这款专门用于 Verilog 和 VHDL 仿真的软件工具，这对于我后续进行数字电路设计和继续学习硬件描述语言提供了帮助。并且通过本次大作业，我还体会到了仿真对于硬件改进设计中具有的实时反馈的重要性。

我参与了部分项目的调试过程，对于在 Quartus II软件编译下载后，在验证过程中出现的错误进行了思考，对于实验箱上出现的真实情况和仿真之间存在的差别有了更为深刻的思考，并且对于如何在错误修改后再次下载进入 FPGA 芯片的过程有了更为充分的认识。当然，在报告撰写的过程中，我也又一次对于在调试过程中未能充分意识到的问题有了新的理解。

通过本次小组大作业，我显示对于本次项目题目中密码箱的原理和结构有了足够的认识，对于如何还原生活中存在的电子器件的结构有了认知。除此以外，在理解编程、仿真模拟、硬件调试方面水平得也到了提升，并通过小组合作锻炼了自己独立思考、团队协作方面的能力。通过合作充分认识到了他人的长处，也

充分认识到当前自己在专业能力方面的不足，通过观察和询问，从小组其他成员那里得到了很多的帮助。相信在本次实训结束后，我的综合素质可以得到各方面的提高，可以对今后的学习和生活中起到更大的帮助。

6.2. 汪江豪

在本次课程中，我深入学习了 Verilog 硬件描述语言和 Quartus II 软件的使用，通过不断学习与纠错的过程，我掌握了实现电子密码锁的底层逻辑，并且明白了它的主体是有限状态机，通过对不同状态间的判断和转移，实现多种模式的切换和功能的实现。此次计算机硬件综合大型作业，我收获良多。

我认识到了 Verilog 等这类硬件描述语言功能的强大，可以通过编写代码，实现数字电路的逻辑功能。并且在调试的过程中，提升了阅读代码的能力。

我更加熟悉了 Quartus II 软件的使用，在 Verilog 代码的编译，创建出对应的逻辑图元件；并进行仿真测试，绑定引脚下载等一系列的过程中，提升了我的动手实践能力和理论联系实际的能力。

我熟悉了 ModelSim 软件的使用。ModelSim 是一款电路仿真软件，它帮助了我对 Verilog 代码进行仿真测试。通过对 testbench 文件的仿真，可以发现程序的漏洞和改进空间。

在本次项目中，我主要参与了程序功能仿真调试以及硬件调试和报告撰写。在此过程中，我深刻意识到，软件设计思路及其重要。它决定了软件的整体结构、功能和性能。只有清晰的设计思路才能帮助我们确定软件的整体结构，即软件各个模块之间的关系和交互方式。这要求我们具备良好的系统分析和设计能力。其次，对程序的调试能力也极为重要，我需要根据软件错综复杂的报错信息，找到问题所在，并尝试修复，这个过程提高了我发现问题，解决问题的能力。并且，此次项目提升了我的自学能力，在初期，无论是 Verilog 语言，还是 VHDL 等 HDL 这类硬件描述语言，我对其都感到很陌生，但通过自己不断地从网上寻找相关资料，视频，初步学习了这门语言，了解了其核心功能，提高了我举一反三的能力，此外，本次课程还使我认识到团队合作的重要性。在项目实施过程中，我们需要分工合作，共同解决问题。这有助于提高我们的团队协作能力和沟通能力。同时，团队合作也可以帮助我们更好地学习和掌握知识，提高学习效果。

本次硬件大作业令我受益匪浅。它不仅锻炼了我的 Verilog 语言编程能力，还提高了我发现问题，解决问题的能力，培养了我的创新精神和团队合作的能力。在科学与技术高速迭代的现代社会，终身学习能力格外重要，因此，本次作业对我今后无论是学习和工作的生涯都具有及其重要的深远影响。

6.3. 何勇乐

本次夏季实训的主题是一个小型密码锁的硬件实现，旨在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，我们能够提高知识应用能力、拓展能力和自主学习能力，并促进产学研相结合。只有拥有分析问题、设计解决方案和实施方案的能力，我们才能将所学的知识真正应用于实际问题解决和创造实际产品的过程中。

在这次实践中，我主要负责小组项目的仿真和调试工作，并参与了少量 Verilog 代码的编写。我们运用所学的数字逻辑知识，并通过融会贯通的方式将其与实际问题相结合。

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 Quartus II 和波形仿真 ModelSim 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，我们将设计的密码锁与生活实际需求相结合，增加了输入位闪烁等额外功能。设计密码锁时，我们使用的是 FSM 有限状态机的思想，在不同输入的情况下进行状态转移，只需要设计好需要哪些状态并为这些状态添加转移条件就完成了整个设计的构造。在仿真时，我们尝试使用编写好的 testbench 文件进行仿真，不仅让仿真结果可复现，而且让仿真的整体流程变的可调控，例如我们一开始并没有正确地为输入进行初始化，导致了一些输出的不稳定情况；后来我们需要额外增加功能，只需要在 testbench 文件中修改增加对应调试内容，针对不同情景修改增加特定不同的输入。这些工具的运用都让我们的整体仿真调试环节变得愉快便捷。

通过这次实践，我们深刻地认识到理论知识在实际生产中的重要性和实际应用方式。我们不仅在解决问题的过程中体会到了知识的实际价值，还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。我们从纯粹在软件上的 Verilog 设计、电路连接、波形仿真模拟，到实际下载 FPGA 运行调试的全过程自主完成，在这期间遇到了许多问题，例如 Quartus II 支持的 Verilog 语言标准不同、Quartus II 下载到 FPGA 芯片上虽然显示成功却并没有真正下载、实验箱哪部分可以产生连续时序、输入位不够的情况下如何缩减、输入好的密码如何保存等问题。这些问题小部分有他人的前车之鉴，但大部分都需要我们自己考虑解决方案和解决方式，例如我们通过循环输入的方式成功解决了输入位不够的情况、通过尝试实验箱的各种端口找到了时序信号产生的位置等等。这些都锻炼了我们发现问题和解决问题的能力，也让我对硬件的设计更感兴趣。

6.4. 周子俊

本次作业中，我负责小组项目大部分的 Verilog 代码设计工作。

显然，作业的需求非常简单，但是考虑到手动布线设计硬件电路难度相当可观，我选择通过 Verilog 语言进行设计工作。于是在夏季学期第一周的前两天，我简单的了解了 Verilog 的语法，并安装 Verilator 等开源工具链进行开发与仿真。

在设计之初，我注意到 Verilog 语言中各语句块类似计算机中的进程，是互相并行的。于是我误以为 Verilog 能够在多个模块驱动单个变量的情况下自动设置锁。在这个前提下，我快速的完成了第一个版本的开发，结果在最终编译时得到报错。这提醒我应该进行充分的准备工作，再进行开发。

在遇到上述问题之后，我想起了数字逻辑课程中学习过的状态机模式。于是我把主程序分为两部分：状态转移部分和显示处理部分，并在一个模块中完成。状态转移部分根据任务要求于时钟信号上升沿时处理状态变化；显示处理部分根据当前状态决定为显示内容添加额外效果或是显示其他内容，使显示能够更加丰富多彩。通过引入合适的设计方式，开发效率得到了很大提升。并且程序功能具有较好的可扩展性。此外，我还通过进行状态的复用，一定程度上减少了代码的浪费。

进行仿真时，小组其他同学告知我结果似乎有错误，而我一开始没能发现错误的真实原因：非阻塞赋值的发生时机和我想象的不相同。由于仿真和实际实验仍然存在差距，不够直观，我们需要结合程序设计的逻辑一步一步分析，才能看出错误之处。这提醒我需要分析语法的实际含义，而不能想当然；并且设计硬件电路需要与实际结合。仿真时，其他同学使用的工具和我不同，导致产生了许多语法差异，进行了很多小的修改。这提醒我需要注意团队合作，否则容易导致效率低下。

不难发现，设计这样的简单硬件电路比我想象的简单一些，是完全能够快速完成的。这和我们学习计算机科学技术的各方面内容类似，需要注重实践，并且不能畏难。结合已有知识，我们能够快速的学会新内容，并且做出实际效果——这也正是计算机科学的魅力。

6.5. 韦梁轩昂

在这次计算机硬件综合大型作业中，我深刻体会到了团队合作的力量和学习新技术的重要性。通过与小组成员的紧密合作，我们成功完成了密码电子锁控制器的设计与实现，这个过程让我对计算机硬件有了更深入的理解，也锻炼了我的编程和问题解决能力。

首先，我对 DICE-SEM 数字模拟综合实验箱的使用更加熟练。DICE-SEM 实验箱是一个功能强大且灵活的教学工具，包含了多种数字和模拟电路模块，能够

模拟各种实际应用场景。在本次作业中，我不仅复习了实验箱的组成和功能，还将这些知识应用到了实际的项目中。例如，在设计密码电子锁控制器时，我们需要使用实验箱中的多种模块来实现复杂的逻辑功能。通过实际操作，我进一步加深了对实验箱硬件的理解，掌握了如何高效地使用这些模块进行项目开发。

FPGA 芯片的可编程性为我们的设计提供了极大的灵活性。FPGA，即现场可编程门阵列，是一种高度可编程的硬件设备，可以通过编写硬件描述语言（HDL）来实现各种复杂的逻辑功能。通过本次作业，我学会了如何在 FPGA 上实现密码电子锁控制器的设计，并且了解了 FPGA 在嵌入式系统、通信和工业控制等领域的广泛应用。这不仅拓宽了我的知识面，也让我看到了 FPGA 在未来工作中的巨大潜力。

在编程方面，Verilog 语言的学习让我受益匪浅。Verilog 是一种硬件描述语言，主要用于数字电路的设计与仿真。在这次作业中，我不仅掌握了 Verilog 的基础语法，还通过实践了解了模块和逻辑块的使用方法。例如，在设计密码电子锁控制器时，我需要编写多个模块来实现不同的功能，如输入密码验证、输出控制信号等。在这个过程中，我学会了如何使用 Verilog 语言进行模块化编程，并且掌握了变量类型和并行执行的限制，这对我的编程能力提升有很大帮助。

在 Quartus II 中运行同伴编写的代码，让我对变量类型和并行执行的限制有了更深的认识。Quartus II 是一款强大的 FPGA 开发工具，提供了从代码编写、仿真测试到硬件下载的一整套解决方案。在这次作业中，我不仅学会了如何使用 Quartus II 进行代码编译和仿真测试，还通过调试同伴的代码，深入理解了 Verilog 语言的各种细节。例如，在调试过程中，我发现了一些代码中的变量类型定义不当的问题，这让我认识到不同类型变量在硬件描述语言中的重要性。

项目的调试过程尤其让我印象深刻。在 Quartus II 软件中编译和下载代码后，我参与了错误的诊断和修正。例如，在实际运行中，我们发现有些功能未能如预期实现，这就需要我们仔细检查代码和硬件连接，找出问题的根源并进行修正。这个过程不仅提高了我的调试技能，也让我对实验箱与仿真之间的差异有了更深的理解。我学会了如何高效地进行硬件调试，以及如何在错误修改后再次下载代码到 FPGA 芯片中。

报告的撰写过程也让我对之前未注意到的问题有了新的认识。撰写报告不仅是对项目成果的总结，也是对整个项目过程的回顾和反思。在撰写过程中，我对每一个设计决策和实现步骤进行了详细的记录和分析，这让我更加全面地理解了项目的各个方面。同时，通过对报告的撰写，我也发现了一些在项目实施过程中未能充分意识到的问题，并提出了改进的建议。

通过这次大作业，我对密码箱的原理和结构有了充分的了解，并且学会了如何还原现实生活中的电子设备。在编程、仿真模拟和硬件调试方面，我的技能得

到了显著提升。这次作业不仅锻炼了我的独立思考和团队协作能力，也让我意识到了自己专业能力上的不足，并从其他成员那里学到了很多。

总的来说，这次硬件大作业对我的专业成长和个人发展都有着不可估量的影响。它不仅提高了我的 Verilog 编程能力，还增强了我的问题发现和解决能力，培养了我的创新精神和团队合作精神。在科技快速发展的今天，终身学习的能力变得越来越重要。我相信，这次作业将对我未来的学习和职业生涯产生深远的影响。通过这次项目的实践，我更加坚定了在计算机硬件领域继续深耕的决心，也为未来的学习和工作奠定了坚实的基础。