北京航空航天大學

BEIHANG UNIVERSITY

实验一、初识实验板——电子八连环

2018 年 10 月 15 日（版本 v01）

网络空间安全学院

目录

实验一、初识实验板——电子八连环（实验指导书部分）

1 实验指导

通过一个实际的数字逻辑电路的实现案例，熟悉 EELAB-FPGACORE2 实验硬件实验平台（以下简称“实验板”）的 FPGA 和串行 PROM 等核心硬件，以及按键、拨码开关、七段数码管等外设；练习通过 Verilog HDL 语言实现 FPGA 可综合代码，以及运用测试程序进行功能仿真的方法；熟悉综合和 FPGA 加载的过程，并实践相应的软硬件调试。

1.1 实验板的硬件资源

实验板核心的 FPGA 芯片的规模达到 50 万门，具有约 1 万个可配置逻辑单元（CLB），360kb（“b”表示比特，“B”表示字节，下同）的块 RAM，以及约 70kb 分布式 RAM。实验板上集成了 JTAG 编程芯片，通过 JTAG 以菊花链的形式连接 PROM 和 FPGA 芯片，可以通过 USB 串口直接编程。

对于实验板编程有两种方式，(1)直接下载*.bit 文件对 FPGA 编程，这种情况下如果掉电则配置丢失；(2)将*.bit 文件转换为*.mcs 镜像文件，对 PROM 编程，形成非易失实现。

在本次实验的基本部分，用到的外设包含 2 路按键、2 路拨码开关、2 位七段数码管，其中用其中一路拨码开关控制工作的模式——“预置”和“求解”；在“预置”模式下，用另一路拨码开关控制预制的高、低位段。

在本次实验的扩展部分，建议试用实验板的 UART（通过 Micro USB）功能，形成具有交互式输入输出的工作状态。

1.2 背景知识——格雷码与九连环

传统玩具九连环（如**图 1**）是按格雷码变化的，如果环在“钗”上为“1”，“板”上为“0”，所谓将环从“钗”上全部取下的操作相当于将某个含有若干个“1”的 9-bit 格雷码比特矢量“倒退”回 `9'b0_0000_0000`。且这种“倒退”是无捷径的，且只

有一个方向可行（即：试图“正向”循环到达 9'b1_0000_0000，然后由 9'b1_0000_0000 返回 9'b0_0000_0000 是行不通的）。

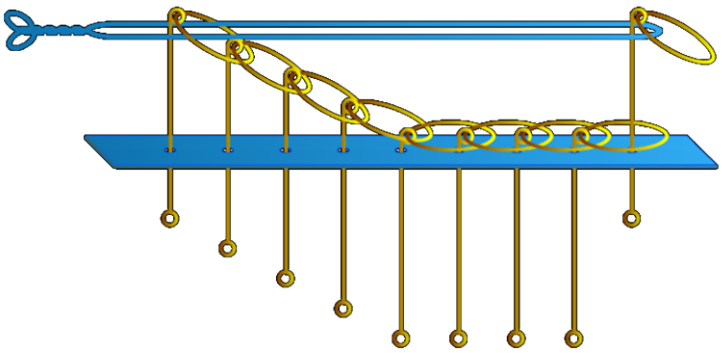

图 1 九连环

利用格雷码到二进制码的转换关系，可以明确解“九连环”的步骤，且可以得到任意一个中间过程（对应一个含有若干个“1”的 9-bit 格雷码比特矢量）距离完全解开的步数。

由于显示资源的限制，本实验板仅适于实现“八连环”的模拟。

1.3 基本实验要求

本实验的目的主要以熟悉和掌握实验板为主，给出了实验的参考模块图（如图 2），学有余力的同学们，可以在基本和扩展要求下，充分发挥设计能力进行改进，或者甚至是全面修正。

基本实验要求为：

a) 通过一路拨码开关控制工作的模式，分为“预置”和“求解”两个模式；

b) 在“预置”模式下，用另一路拨码开关控制预制的高、低位段；

c) 在“预置”模式下，通过两个按键分别以“加”、“减”计数改变当前位段的值，且以 4 位 LED 灯显示该比特矢量；

d) 在“预置”模式下，相应位段数码管根据预制的值即时显示 0~9，以及 A、b、C、d、E、F 字符；

e) 在“求解”模式下，每按动一个按键一次，则对应运行解环步骤一次，这样可以一步一步解开八连环；

f) 在“求解”模式下，如果八连环已经转换到 8'b0000_0000，则停下不动；

g) 当高 4 位和低 4 位分别输入的时候，如果切换拨码开关后，另一部分的存储值要保持为切换前的值。（提示：或许图 2 中的控制结构需要改进）

请注意，所谓的思考和改进在一定程度上并没有唯一解，往往是多种解决方案的权衡和比较。

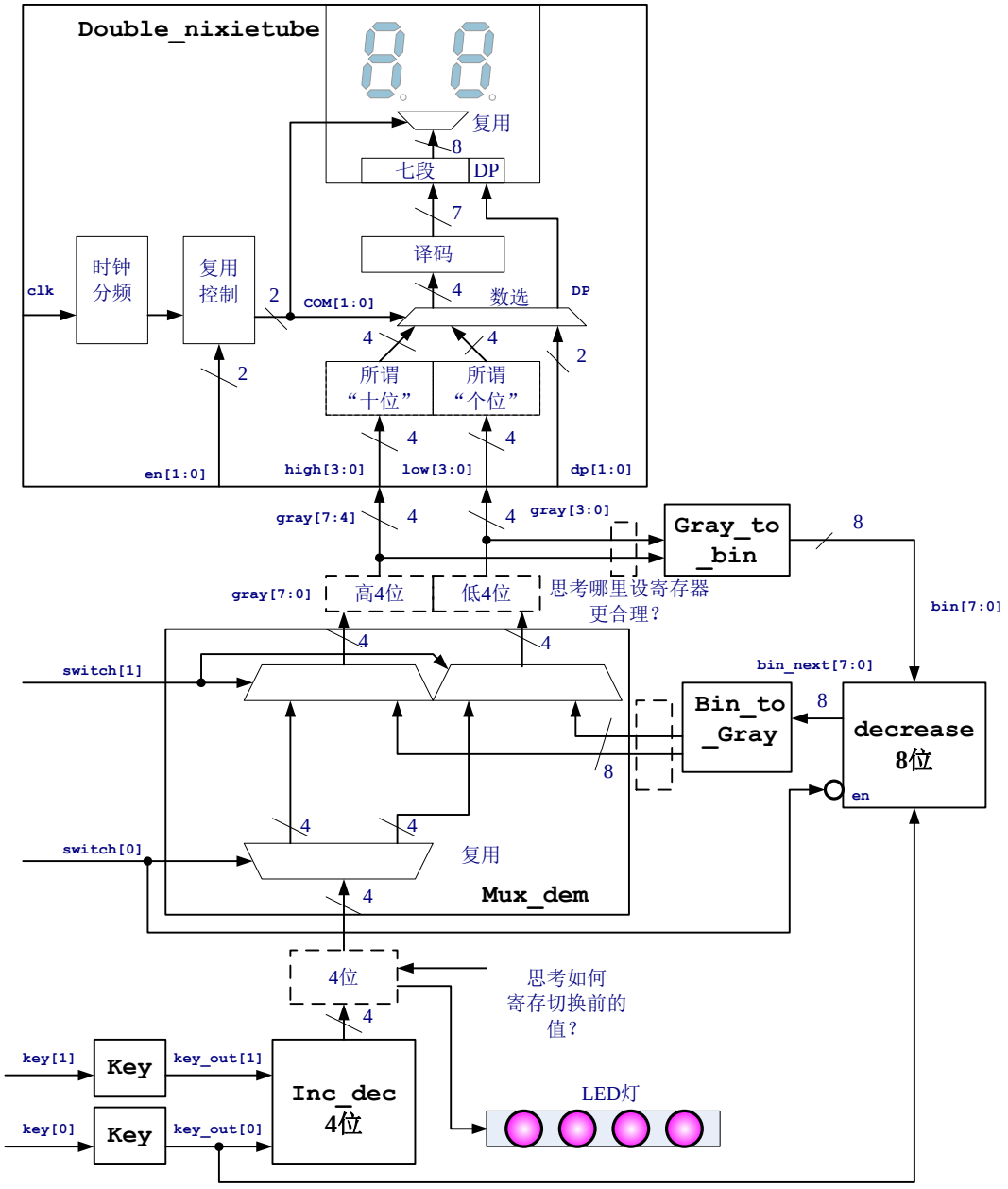

图 2 参考模块图（基本，待改进）

由于本实验阶段为“初识”，因此仅以实验报告为考核形式，具体**截止期限**请注意任课教师的通知，实验报告格式请参考本文档。

后续的实验将采取**实验报告和现场汇报演示相结合**的形式，特此广而告之。

1.4 提示和扩展要求

提示：在程序实现中，请权衡代码的封装和复杂性的代价，灵活地或者以模块实现，或者以连续赋值或过程块实现。

扩展的实验要求（根据完成的程度可酌情提高评价的等级）

h) 在“求解”模式下，用另一路拨码开关控制显示的方式，可以显示格雷码，也可以显示自然二进制码；

i) 当从“预置”模式切换到“求解”模式的时候，预制的 8 位比特矢量不变（注意：但要注意另一路拨码开关的位置，可以以格雷码显示，也可能以自然二进制码显示，当以自然二进制码显示的时候，数码管的显示内容可能有变化）；

j) 当从“求解”模式切换到“预置”模式的时候，数码管自动转换为格雷码（如果已经用格雷码显示，则不变）；

k) 在“求解”模式下，另设一个按键，按一次则按照每 1 秒一步的速度解环，对应数码管即时显示，直到再按一次该按键停止；

l) 利用 Micro USB 构建的 UART，形成交互状态的“求解”，即用户输入下次预期的环的位置，由 FPGA 判断是否可能（注：由“八连环”的物理结构限制可以由当前状态一步到达，但这也可能不是向正确的方向求解），如果可能则改变电路中 8 位比特矢量，并通过 UART 反显，否则通过 UART 提示出错。

1.5 已有软件资源

1.5.1 时钟分频

实验板的主频为 50MHz 时钟，过快的时钟会过多耗能，因此设置了时钟分频的过程块，可供改写使用。

1.5.2 数码管

两个数码管采用动态刷新，需要用寄存器将它们显示的内容寄存起来，通过选通端 COM[1:0]，交替刷新显示。

1.5.3 按键去抖

按键需要去抖，提供按键去抖模块 Key 可供利用。

1.5.4 已有程序框架

为了便于入门，提供“预置”模式下按键“加”、“减”计数及点亮 4 个 LED 灯的程序。

数码管显示也提供模块 Nixietube，但部分字模不全（即：A、b、C、d、E、F），请补充。

1.5.5 引脚和 I/O 命名

附录 A 提供实验板 FPGA 引脚速查，请结合《EELAB-FPGACORE2 使用简介》查对；附录 B 提供一个可供参考的 UCF 定义，Verilog HDL 编程时可参考其 I/O 变量命名。

1.5.6 实验板的 UART

提供一个只能每次读写一个字节的 UART 的样例源代码，具有固定的 9600 波特率、8 位数据位、偶校验、1 位停止位。更好的 UART（注：更稳定可靠、波特率可配置、有 Telnet 等上层协议）需要用户自行开发。

1.5.7 实验板的其它资源

具有 12 位的 ADC 芯片 ADS7951，由于它和 FPGA 采用 SPI 串行接口，写控制指令和读取数据都要根据 SPI 总线的协议，在实验中心给出的例子程序中具有 4 位 ADC 测量的样例。（但由于注释较少，可读性较差）

在实验板上，ADC 的输入范围是 0 到 2.5 伏，请注意使用，避免短路或烧毁。

用户 I/O 资源可以根据引脚定义进行实验，请注意电平的范围，避免短路或击穿。

实验一、初识实验板——电子八连环（实验报告部分）

2 实验报告

2.1 实验背景与需求分析

2.1.1 实验背景

九连环是中国传统民间智力玩具，以金属丝制成 9 个圆环，将圆环套装在横板或各式框架上，并贯以环柄。九连环的每个环互相制约，只有第一环能够自由上下。要想取下或置上第 n 个环，就必须满足两个条件（第一个环除外）：

（1）第 n-1 个环在架上；

（2）第 n-1 个环前面的环全部不在架上。

九连环中的每个环都有上下两种状态，如果把这两种状态用 0/1 来表示的话，这个状态序列就会形成一种循环二进制编码（格雷码）的序列。即将九连环抽象为一个数学模型格雷码，然后将解九连环的过程抽象为任意数值的格雷码一步步变为 0 的过程。实验仿照九连环的解法，通过编程在 FPGA 上实现“电子八连环”的设置和求解。

2.1.2 需求分析

实验要求分为两个部分：预置和求解。

预置即人为设置八连环的初始状态，并且通过数码管将当前状态的格雷码显示出来，同时 LED 显示比特矢量。

求解即从预置设置的状态一步步求解，当数码管显示 00 时，表明求解完成，求解时可以通过拨码开关设置显示格式是格雷码还是自然二进制码。

2.2 系统设计

2.2.1 总体设计思路

我们的系统主要分为四个模块，分别是顶层模块，按键去抖模块，数码管显示模块和 LED 显示模块。顶层模块中在逻辑上可分为预置部分和求解部分，由于两个操作

需要控制同一个寄存器，为了避免竞争冒险，我们都将其设置在顶层模块中，用八位寄存器 gray 来保存格雷码，八位寄存器 num 保存数码管要显示的值。

在每个系统时钟的上升沿，预置部分会根据拨码开关和按键输入来改变寄存器 gray 的内容，而求解部分会根据 gray 中存储的格雷码计算自然二进制码，将其存储于 num 寄存器中，如果此时需要求解，则进行一步求解操作，最后根据多选器来决定如何更新寄存器 gray 和 num 的内容。

由于按键去抖，数码管显示和 LED 显示功能均不需要更新 gray 寄存器的值，所以我们可以将其分为单独的模块，通过接口分配来模块化它们的功能。系统的整体设计流程图如下：

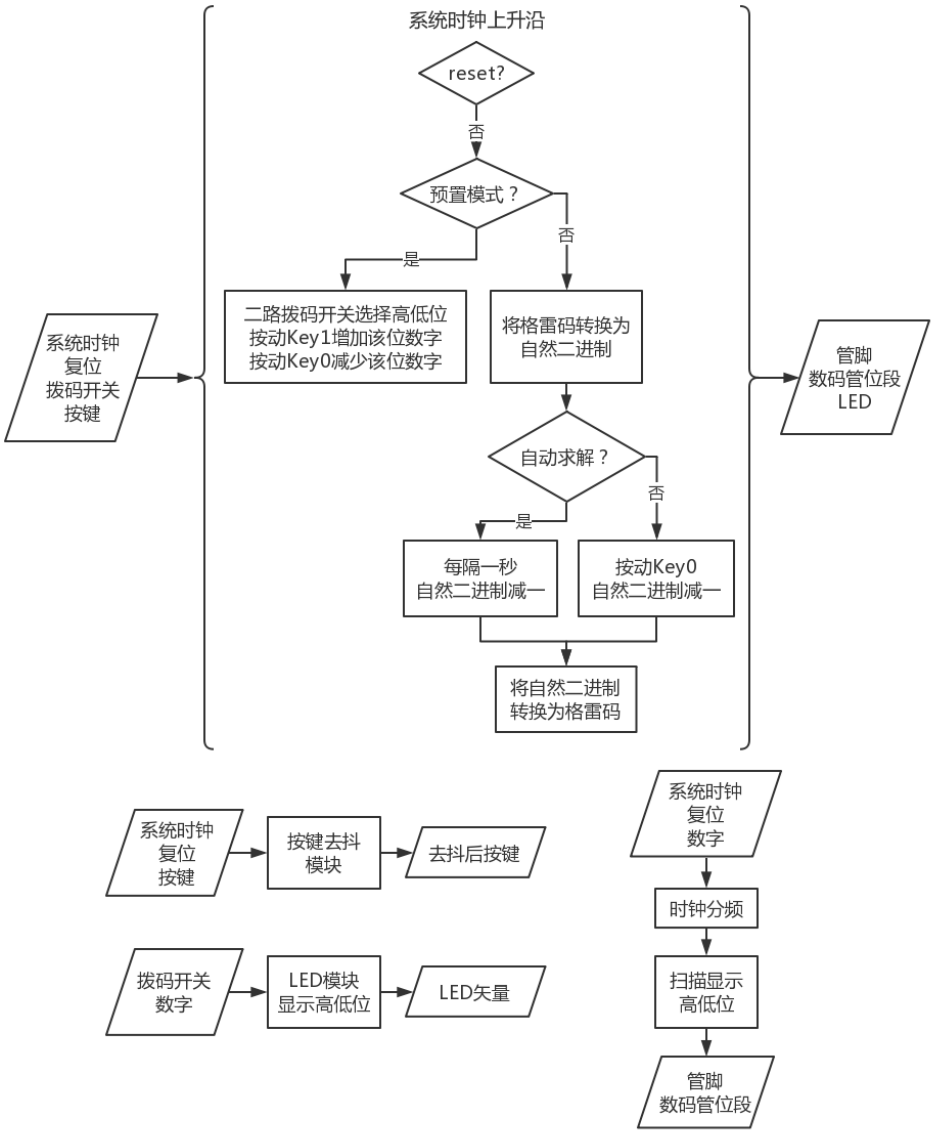

图 3 系统设计图

2.2.2 接口设计

实验板提供的 IO 分别有系统时钟，复位，拨码开关，按键，数码管管脚，选通端和 LED 灯，这些接口必须先在顶层模块中定义，由于按键去抖模块和数码管显示模块均需要时钟信号控制，所以也将系统时钟作为他们的输入。除此之外，按键去抖模块还需要按键信号输入，输出去抖后的按键信号；数码管模块以寄存器 num 作为输入，输出数码管的显示信号；LED 显示模块以拨码开关和寄存器 num 作为输入，输出 LED 的显示信号，同时也在必要的时候将复位信号分配给相应的模块。

2.2.3 预置模块

首先由第二路拨码开关 switch[0]来判断现在处于高四位还是低四位，之后通过两个按键对相应的位进行加一或减一的操作。由于预置时数码管显示的是格雷码，所以要将寄存器 gray 的值存入将要显示的 num 寄存器中。

```
if (switch[0] == 1'b0) begin // 控制低四位
   if (key_out[1] == 1'b1) // 加一
      gray[3:0] = gray[3:0] + 1;
   else if (key_out[0] == 1'b1) // 减一
      gray[3:0] = gray[3:0] - 1;
end
else begin // 控制高四位
   if (key_out[1] == 1'b1) // 加一
      gray[7:4] = gray[7:4] + 1;
   else if (key_out[0] == 1'b1) // 减一
      gray[7:4] = gray[7:4] - 1;
end
num = gray;
```

2.2.4 求解模块

求解模块的主要设计思路是用 8 位寄存器 gray 来储存格雷码，8 位寄存器 num 来储存自然二进制码，首先通过以下程序将格雷码转换为自然二进制码：

```
num[7] = gray[7];
for (i = 6; i >= 0; i = i - 1)
   num[i] = num[i+1] ^ gray[i];
```

每次求解过后，我们需要将自然二进制码再转回格雷码，更新寄存器的内容，通过以下程序将自然二进制码转换为格雷码：

```
gray = (num >> 1) ^ num;
```

每当按下手动求解按键求解时，首先通过检验寄存器 gray 的值是否为 0 判断是否达到最终状态，若还有下一步可解，则将自然二进制码减 1，然后将其转换为格雷码，更新寄存器内容，以下程序为手动求解的部分：

```
if ((key_out[0] == 1 && gray != 8'h00)) begin //key[0]为手动求解
   num = num - 1;
   gray = (num >> 1) ^ num;
end
```

若要实现自动求解，则基本代码无需改变，只需要添加计时器并且判断是否在自动求解的状态变量即可，以下程序为自动求解的部分：

```
parameter solve_interval = 50000000; // 确保间隔为一秒
else if (key_out[1] == 1)  // key[1]为自动求解
   auto_solve = ~auto_solve; // 每次取先前状态的反
else begin
   if (auto_solve == 1) begin // auto solve
      cnt = cnt + 1;
      if (cnt == solve_interval && gray != 8'h00) begin
         cnt = 0;
         num = num - 1;
         gray = (num >> 1) ^ num;
      end
   end
end
```

为了实现扩展功能，即用第二路拨码开关控制数码管以格雷码还是自然二进制码显示数字。由于寄存器 num 始终保存着自然二进制码的值，而数码管显示模块以 num 为输入，若输出为格雷码格式，只需要将格雷码赋值给 num 寄存器即可：

```
if (switch[0] == 0)
   num = gray;
```

2.2.5 数码管及 LED 显示模块

由于 FPGA 上有两位数码管，我们采用动态扫描刷新，让两个数码管交替刷新显示数字，该模块以需要显示的数字，以及系统时钟和复位作为输入，并且输出数码管的管脚和位段。

首先建立时钟分频模块，新的时钟间隔为 0.02 秒：

```
parameter update_interval = 50000000 / 200 - 1;
always @(posedge clk or negedge rst) begin
   if (!rst) begin
      cnt <= 0;
```

```
            sel <= 0;
        end
        else begin
            cnt <= cnt + 1;
            if (cnt == update_interval) begin
                cnt <= 0;
                sel <= ~sel;
            end
        end
    end
```

然后在每次时钟信号改变时，切换选通端，并且将输入数据的高 4 位或者低 4 位存储在待显示数字的寄存器中，此时数码管刷新频率为 50Hz，满足要求：

```
    always @(*) begin
        case (sel)
            1'b0: begin dat <= number[7:4]; com <= 2'b01; end
            1'b1: begin dat <= number[3:0]; com <= 2'b10; end
        endcase
    end
```

最后通过控制管脚的信号让相应数码管显示数字即可：

```
    always @(dat) begin
        seg[0] <= 1'b0;
        case (dat)
            4'h0: seg[7:1] <= 7'b1111110;
            4'h1: seg[7:1] <= 7'b0110000;
            4'h2: seg[7:1] <= 7'b1101101;
            4'h3: seg[7:1] <= 7'b1111001;
            4'h4: seg[7:1] <= 7'b0110011;
            4'h5: seg[7:1] <= 7'b1011011;
            4'h6: seg[7:1] <= 7'b1011111;
            4'h7: seg[7:1] <= 7'b1110000;
            4'h8: seg[7:1] <= 7'b1111111;
            4'h9: seg[7:1] <= 7'b1111011;
            4'hA: seg[7:1] <= 7'b1110111;
            4'hB: seg[7:1] <= 7'b0011111;
            4'hC: seg[7:1] <= 7'b1001110;
            4'hD: seg[7:1] <= 7'b0111101;
            4'hE: seg[7:1] <= 7'b1001111;
            4'hF: seg[7:1] <= 7'b1000111;
            default: seg[7:1] <= 7'b0000000;
        endcase
    end
```

根据实验要求，在预置模式四位 LED 显示当前位段的比特矢量，而求解模式 LED

不显示内容，我们建立两个多选器 MUX 即可实现所需功能：

```
always @(*)
       if (switch[1] == 0) // 预置模式
          if (switch[0] == 0) // 低四位
             out <= number[3:0];
          else // 高四位
             out <= number[7:4];
       else // 求解模式
          out <= 4'h0;
```

2.3 功能仿真测试

2.3.1 测试程序设计

我们建立了 Top 模块的 test bench，利用 ISE 提供的 ISim 工作台来仿真系统的工作方式，由于存在按键去抖模块，我们需要对按键的高电平延迟一段时间，最后通过观察信号即可分析系统的工作状态。以下为测试程序的主体内容：

```
initial begin
   // 初始化输入
   sys_clk = 0;
   sys_rst = 1;
   switch = 2'b00;
   key = 2'b00;
   // 延迟 100ns 以等待全局复位完成
   #100;
   // 添加激励信号
   sys_rst = 0;
   #10;
   sys_rst = 1;
   #10;
   switch = 2'b00; // 切换为预置模式（低位）
   key = 2'b10; // 加一
   #2000000;
   key = 2'b00;
   #2000000;
   key = 2'b10; // 加一
   #2000000;
   key = 2'b00;
   #2000000;
   switch = 2'b01; // 切换为求解模式
   #10;
```

```
        key = 2'b01; // 求解一步
        #2000000;
        key = 2'b00;
        #2000000;
        key = 2'b01; // 求解一步
        #2000000;
        key = 2'b00;
    end
    always #1 sys_clk = ~sys_clk; // 模拟系统时钟
```

2.3.2 功能仿真过程

通过 ISim 得到的仿真波形图如下：

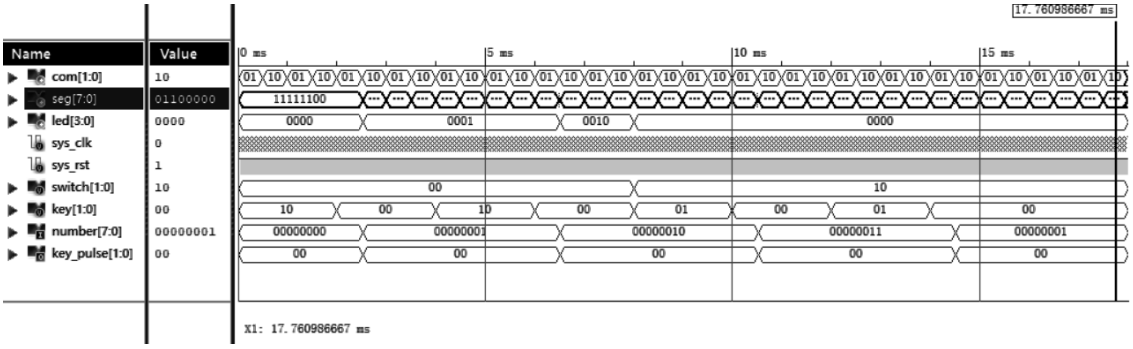

图 4 仿真波形图

2.3.2 实验关键结果及其解释

观察波形图，可以看到在预置时按动两次按键，寄存器 number 的值由 0 变为 2，在求解过程，寄存器 number 的值按照 2→3→1 的方式改变，与格雷码递减方式 10→11→01 相同，说明系统工作正常。

2.4 设计实现

2.4.1 综合和下载过程

语法检验通过后，通过 PlanAhead 按照附录 A 中的说明进行引脚分配。将系统用到的 8 个数码管引脚，2 个数码管片选信号引脚，2 个按键信号引脚，4 个 LED 引脚，2 个拨码开关引脚和时钟信号、复位信号绑定到实验板上，分配好之后生成 Top.ucf 文件。

Name	Direction	Neg Diff Pair	Site	Fixed
All ports (20)				
com (2)	Output			
com[1]	Output		P27	☑
com[0]	Output		P26	☑
key (2)	Input			
led (4)	Output			
seg (8)	Output			
switch (2)	Input			
Scalar ports (2)				

图 5 引脚分配

然后在 ISE 中进行综合，观察电路原理图：

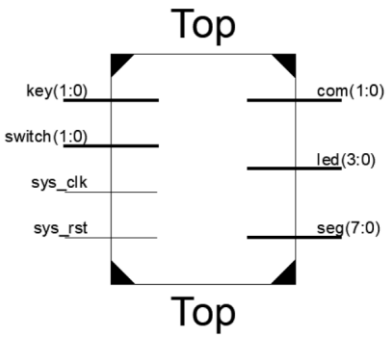

图 6 电路原理图

最后在 iMPACT 中将程序烧录到实验板中：

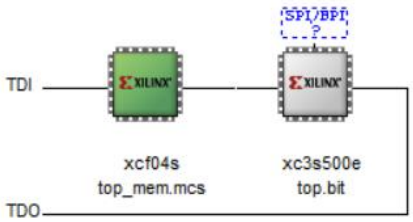

图 7 程序烧录

2.4.2 实验关键结果及其解释

接通电源后，在第一路拨码开关为“0”时，为预制模式，这时通过第二路拨码开关选择高四位或者低四位进行操作，操作时四位 LED 显示当前操作数位的二进制比特矢量。通过两个按键分别进行加减设定起始数字后，将第一路拨码开关打到“1”，这时进入求解模式，第二路拨码开关为“1”时显示格雷码，为“0”时显示自然二进制码。按下按键 1，进入自动求解模式，按下按键 0，手动求解模式求解一步。

2.5 小结

通过此次实验，我们学习到的知识有：

1. 九连环与格雷码的联系；
2. 初步熟悉了 FPGA 开发板；
3. ISE 和 Modelsim 的基本操作，HDL 的综合和模拟；
4. 硬件操作语言 Verilog 的基本语法以及各种模块的实现方法；
5. 深入理解了书中的一些概念。比如电平触发和时钟边沿触发。

我们的以下能力得到了提升：

1. 对 Verilog 中自顶而下的模块化设计方法的理解；
2. 从 demo 中学习编程语言的能力。即从实践中研究问题、分析问题和解决问题的能力；
3. 动手能力。

本次实验所用代码已经上传到 Github 仓库：

https://github.com/hiyouga/digiC-experiment/tree/master/ex1

附录 A 实验板 FPGA 引脚速查表

FPGA 引脚序号	原理图引脚定义	引脚说明	备注（参考命名）
P36	“Sys_CLK”	系统时钟	主频：50MHz
P26	COM1	“位”选信号——十位	**"COM[0]"**
P27	COM2	“位”选信号——个位	**"COM[1]"**
P23	LED_A	“段”选信号——A，共阴极	**"SEG[7]"**
P18	LED_B	“段”选信号——B，共阴极	**"SEG[6]"**
P15	LED_C	“段”选信号——C，共阴极	**"SEG[5]"**
P16	LED_D	“段”选信号——D，共阴极	**"SEG[4]"**
P17	LED_E	“段”选信号——E，共阴极	**"SEG[3]"**
P22	LED_F	“段”选信号——F，共阴极	**"SEG[2]"**
P24	LED_G	“段”选信号——G，共阴极	**"SEG[1]"**
P12	LED_DP	“段”选信号——小数点，共阴极	**"SEG[0]"**
P32	SW1	拨码开关第 1 路	**"Switch[0]"**
P33	SW2	拨码开关第 2 路	**"Switch[1]"**
P79	LED0	LED 第 1 路	**"LED[0]"**
P83	LED1	LED 第 2 路	**"LED[1]"**
P84	LED2	LED 第 3 路	**"LED[2]"**
P85	LED3	LED 第 4 路	**"LED[3]"**
P62	CS	片选输入，以 ADS7951 芯片为判断方向	**"AD_CS"**
P58	SDO	ADC 串行数据输出， 以 ADS7951 芯片为判断方向	**"AD_SDO"**
P60	SDI	ADC 串行数据输入， 以 ADS7951 芯片为判断方向	**"AD_SDI"**
P61	SCLK	串行时钟输入， 以 ADS7951 芯片为判断方向	**"AD_SCLK"**
P2	KEY0	按键 0	**"Key[0]"**
P3	KEY1	按键 1	**"Key[1]"**
P11	RESET	复位按键	上拉,**"Sys_RST"**
P9	TX	USB 串口发送， 以核心 FPGA 为判断方向	NET **"Uart_Rx"** LOC = P9;
P10	RX	USB 串口接收， 以核心 FPGA 为判断方向	NET **"Uart_Tx"** LOC = P10;
P63	D0	可扩展 IO_0	电路板正面
P65	D1	可扩展 IO_1	同上
P66	D2	可扩展 IO_2	同上
P67	D3	可扩展 IO_3	同上
P68	D4	可扩展 IO_4	同上
P70	D5	可扩展 IO_5	同上
P71	D6	可扩展 IO_6	同上

P78	D7	可扩展 IO_7	同上
P86	D8	可扩展 IO_8	同上
P90	D9	可扩展 IO_9	同上
P91	D10	可扩展 IO_10	同上
P92	D11	可扩展 IO_11	同上
P94	D12	可扩展 IO_12	同上
P95	D13	可扩展 IO_13	同上
P98	D14	可扩展 IO_14	同上
P99	D15	可扩展 IO_15	同上
P34	D16	可扩展 IO_16	同上

说明：

- ADC 采用 SPI 串行数据总线读入读出数据，最高 SPI 速度可以被配置为 20MHz，ADC 采样率为 1MHz。

附录 B UCF 文件中的定义（Verilog HDL 编程时参考的 I/O 变量命名）

```
NET "Sys_CLK" LOC = P36;    /* EEBUAA 实验板的主频为 50MHz */
NET "SEG[0]" LOC = P12;     /* LED_DP 数码管的小数点 */ /* “数码管”的英文为 nixietube*/
NET "SEG[1]" LOC = P24;     /* LED_G */
NET "SEG[2]" LOC = P22;     /* LED_F */
NET "SEG[3]" LOC = P17;     /* LED_E */
NET "SEG[4]" LOC = P16;     /* LED_D */
NET "SEG[5]" LOC = P15;     /* LED_C */
NET "SEG[6]" LOC = P18;     /* LED_B */
NET "SEG[7]" LOC = P23;     /* LED_A */
NET "COM[0]" LOC = P26;     /* “位”选信号 COM1，“十”位 */
NET "COM[1]" LOC = P27;     /* “位”选信号 COM1，“个”位 */
NET "Switch[0]" LOC = P32;     /* 拨码开关第 1 路 */
NET "Switch[1]" LOC = P33;     /* 拨码开关第 2 路 */
NET "LED[0]" LOC = P79;     /* LED0 */
NET "LED[1]" LOC = P83;     /* LED1 */
NET "LED[2]" LOC = P84;     /* LED2 */
NET "LED[3]" LOC = P85;     /* LED3 */
NET "AD_CS" LOC = P62;      /* CS */
NET "AD_SDO" LOC = P58;     /* SDO */
NET "AD_SDI" LOC = P60;     /* SDI */
NET "AD_SCLK" LOC = P61;    /* SCLK */
NET "Key[0]" LOC = P2;      /* KEY0 */
NET "Key[1]" LOC = P3;      /* KEY1 */
NET "Sys_RST" LOC = P11;    /* RESET */
NET "Sys_RST" PULLUP;
NET "Uart_Tx" LOC = P10;    /* Uart_Tx 是以外部设备为判断方向，RX 以本地设备为判断方向 */
NET "Uart_Rx" LOC = P9;     /* Uart_Rx 是以外部设备为判断方向，TX 以本地设备为判断方向 */
```