3 | 时序逻辑入门

记忆的永恒（萨尔瓦多 • 达利）

时间是不断运动的永恒影像。

~ · ~

3.1 | 实验目的

本实验的目的是学习时序逻辑模块（状态机）在数字系统中的应用；掌握实验平台的外部功能模块在数字系统设计中的应用。

3.2 | 实验内容

3.2.1 | 流水灯

Ego1 上有 16 个 LED 灯，我们可以使用这些 LED 灯实现许多功能。其中，流水灯便是一个最简单的应用。你需要使用已经给出的计数器 `counter.v`、控制模块 `flash_led_ctl.v`、顶层文件 `flash_led_top.v` 以及测试激励文件 `Flash_led_top_tb.v`, 在 Ego1 上实现一个流水灯，其移动速度（即每秒流动次数）由如下公式计算：

$$\text{移动速度} = \frac{\text{处于高电平的拨码开关数量} - 4}{8}$$

注意，移动速度为负数时，流水灯的方向需要切换

关于计时器的详细说明见附录。

3.2.2 | 智能报警器

设计并实现一个智能报警器，使用按键触发报警（按键消抖教程见附录），LED 灯进行状态指示，并通过报警次数的增加来调整闪烁频率。同时，用户可以通过拨码开关手动调节闪烁频率倍数，具体要求如下：

1. 按下按键 B1 后，触发 “警报”（点亮 LED 灯 L1 ），并保持；
2. 直到再次按键 B1 后，解除警报；
3. 发生警报时，让 L1 闪烁；

4. 统计触发警报的次数，并将次数在数码管上显示；
5. 按下按键 B2 后，仍保持当前的模式，但报警次数变为 1；
6. 在不同警报次数下，让 L1 闪烁的频率不同，次数越多、闪烁越快；
7. 拨码开关 K1 ，K2 也可改变闪烁频率，设此时闪烁频率 N，打开开关 K1 时，闪烁频率变为 $2N$，打开 K2 时变为 $4N$。

3.2.3 | 密码锁

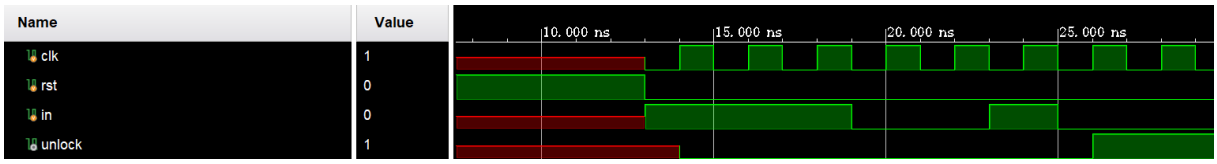
有个密码锁，只有输入 `0010` 的时候才能成功进行解锁，

例如，如果输入 `11001` 的时候不会解锁，接着输入 `0`，便能解锁。一旦开锁后，不再进行检测，unlock 维持高电平。

现在我们希望通过数字电路来实现判断密码锁能否解锁。给出以下模块端口定义：

```
1  module lock(
2      input      clk,     // clock
3      input      rst,     // reset
4      input      in ,
5      output reg unlock
6  );
```

根据给出模块实现要求功能。所期望的波形示意图如下所示：



密码锁无需上版验证，只需仿真即可。

3.3 | 实验要求

1. 上版流水灯、分析代码，并画出代码结构图；
2. 写出智能报警器中调节闪烁速度的实现思路；
3. 画出密码锁的状态转移图；
4. 在实验报告中提交系统级设计模块图、设计代码、激励程序（不必须包含所有模块的）、仿真波形结果截图（与激励配套）、板级实测验证结果照片，其中，系统级设计模块图要求给出整个系统的数据输出信号，系统内各个子模块的输入输出信号和模块间的连接关系；
5. 提交实验报告和所有源程序文件的压缩包。

C | Ego1 clk 与时钟分频

C.1 | Ego1 clk

时钟（clock，一般写为 clk）是数字逻辑的基石。没有时钟就没有时序电路。因此，认识时钟为何物以及我们为何需要时钟是非常重要的。虽然可以通过手动拉高、拉低输入的方式提供上升与下降沿，以达到时钟的效果，但每秒拨动 10^8 次拨码开关会对人和设备都造成难以接受的损伤。所以说，一个稳定时钟源是必要的。

Ego1 提供了一个 100MHz 的时钟，位于 P17 引脚上。他负责向 U1 的 FPGA 芯片提供稳定的时钟源。对于一个时钟来说，我们现在需要关心的有两个参数：主频和占空比。主频即此处提到的 100MHz，意味着这个时钟在 1s 内产生的脉冲数。占空比指时钟上升沿占一个时钟周期（即频率的倒数）的比例。只有理解了主频和占空比，之后在实现时序逻辑电路时，才能得心应手，如鱼得水。

C.2 | 改变时钟频率

我们没有办法更改 Ego1 100MHz 的时钟源。倘若我们需要一些较为低频的时钟，就需要分频器的介入。以下模块描述了一个时钟分频器：

```
module counter(
    input  wire clk,
    input  wire rst,
    output wire clk_bps
);
    reg [13: 0] cnt_first, cnt_second;

    always @(posedge clk)
        if(rst)
            cnt_first <= 14'd0;
        else if(cnt_first == 14'd10000)
            cnt_first <= 14'd0;
        else
            cnt_first <= cnt_first + 1'b1;

    always @(posedge clk)
        if(rst)
            cnt_second <= 14'd0;
        else if(cnt_second == 14'd10000)
            cnt_second <= 14'd0;
        else if(cnt_first == 14'd10000)
            cnt_second <= cnt_second + 1'b1;

    assign clk_bps = cnt_second == 14'd10000 ? 1'b1 : 1'b0;
endmodule
```

该模块接收两个输入：时钟信号 `clk` 与复位信号（reset，简写为 rst）`rst`；并且输出一个脉冲时钟信号 `clk_bps`。该模块定义了两个计数器，`cnt_first` 与 `cnt_second`，负责计数工作。每过一个时钟周期都会使某一个或两个计数器自增，直到达到特定条件两个计数器均会清零。此时将会拉高脉冲时钟信号 `clk_bps` 一个周期。我们可以认为，`clk_bps` 的频率是计数器两次达到

特定条件的时间间隔之倒数，其占空比将随频率变化，但高电平的持续时间永远为一个时钟周期的长度。

利用上述的模块，我们便可以通过高频时钟生成一个或多个频率较低的时钟，以满足我们的实际需要。

如果需要了解更多内容,请访问https://ustb-806.github.io/DigitalLogic_Info/#/module/running_led

D | 按键消抖与 Ego1 外设说明

D.1 | EG01 按键说明

EGO1 具有 2 个专用按键和 5 个通用按键。本实验中主要用到 5 个通用按键，在 PCB 板上的名称是 S0，S1，S2，S3，S4。

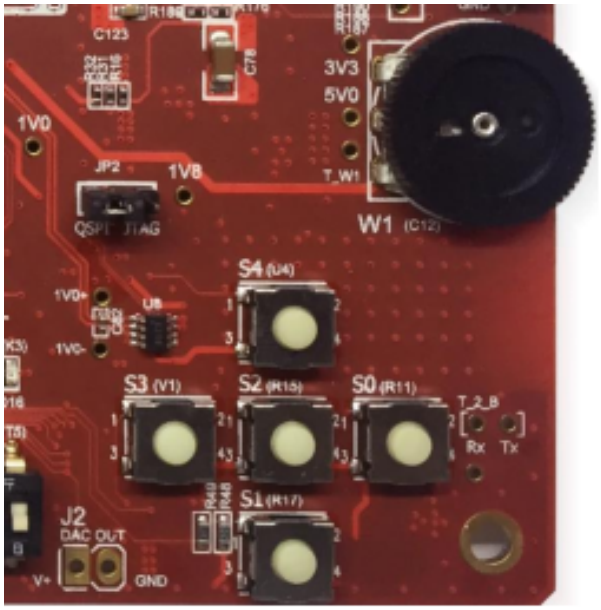


图 D.1: Ego1 通用按键

下图以 S0 为例显示了通用按键的电路原理图，图中按键输出信号 PB0 连接到了 PFGA 的 R11 管脚（提示：电路原理图中，通常以同一个名称来命名多个信号线，表示这几个信号线都是同一个根信号线，这样使得复杂的原理图变简洁）。从该原理图可知：

1. 当 S0 按键没有按下时，PB0 低电平；
2. 当 S0 被按下时，PB0 为高电平；

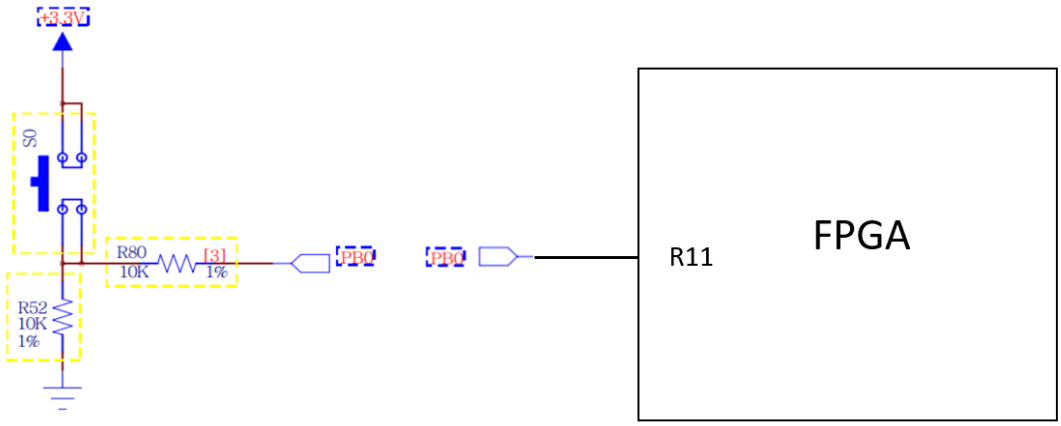


图 D.2: 按键原理图

管脚约束见手册。

D.2 | 按键防抖说明

为了保证键每闭合一次 FPGA 仅作一次处理，必须去除键按下时和释放时的抖动。开发板使用的按键是触点式的，由于按键是机械触点，当机械触点断开、闭合时，会有抖动。

图 D.3: 按键抖动示意图

按键抖动示意图中的这种抖动对于人来说是感觉不到的，但对于 FPGA 或者处理器来说，其运行速度是在微秒级甚至纳秒级，而机械抖动的时间至少是毫秒级，因此这种抖动是一个“漫长”的时间。

对于按键存在的抖动，如果在抖动过程中高低电平的状态没发生变化，则这个抖动我们是不需要考虑的。但是，如果在抖动过程中高低电平状态发生了变化甚至是频繁变化，则这个抖动我们就消除，下面所提到的抖动就是这种类型的抖动。为使 FPGA 能正确的读出按键的状态，对每一次按键只作一次响应，就必须考虑如何去除抖动，常用的去抖动的方法有两种：硬件方法和软件方法。

FPGA 设计中，常用软件法去抖，因此对于硬件方法我们在此不作介绍。软件法去抖其实很简单，按键初始状态为低电平，当 FPGA 获得键值为 1 的信息后，不是立即认定按键已被按下，而是延时 5ms 或更长一些时间后再次检测按键，如果仍为低，说明按键的确按下了，这实际上是避开了按键按下时的抖动时间。而在检测到按键释放后再延时 5ms，消除后沿的抖动，然后再对键值处理。当然，实际应用中，按键的质量也是千差万别，要根据按键的不同，来设定这个延时时间，通常这个延时时间不会太短，一般设为 5~20ms。具体做法是，将按键信息延时 3 次取样 3 次，每次延迟取样间隔约为 5ms，当这 3 个取样值都一样时，说明抖动已消失，如果 3 个取样值不一样，说明抖动存在，直至这 3 个取样值一样时，才认为按键稳定。将这 3 个取样值与运算后得到的信号作为按键的状态，这个按键状态可作为稳定的按键输入，参与到后续对按键的处理操作。

D.3 | 拨码开关说明

EGO1 上有 8 个拨码开关（SW0~SW7）

图 D.4: SW0~SW7

以 SW0 为例，拨码开关的原理图如下图所示，开关输出管脚 SW_0 与 FPGA 的 P5 管脚相连。开关具有三个状态：

1. 开关拨到 3.3V 侧，SW_0 为高电平；
2. 开关拨到 GND 侧，SW_0 为低电平；

3. 开关拨到中间，SW_0 位悬空，建议不要用悬空。

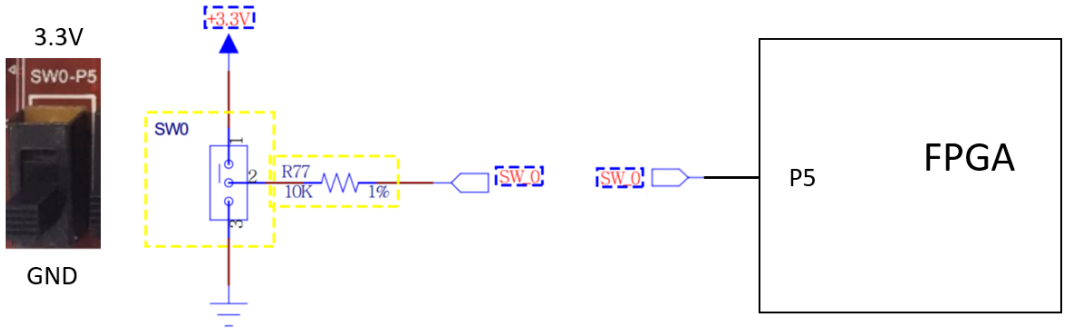


图 D.5: 拨码开关原理图

管脚约束见手册。

D.4 | DIP 开关说明

1 个 8 位的 DIP 开关（SW8, 包含 8 个小开关），如下图所示。

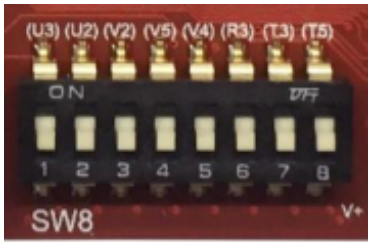


图 D.6: DIP 开关 SW8

DIP 拨码开关的原理图如下图所示，开关具有两种状态。以 SW_DIP0 为例

1. 开关拨到“ON”侧，SW_DIP0 为高电平；
2. 开关拨到“1”侧，SW_DIP0 为低电平；

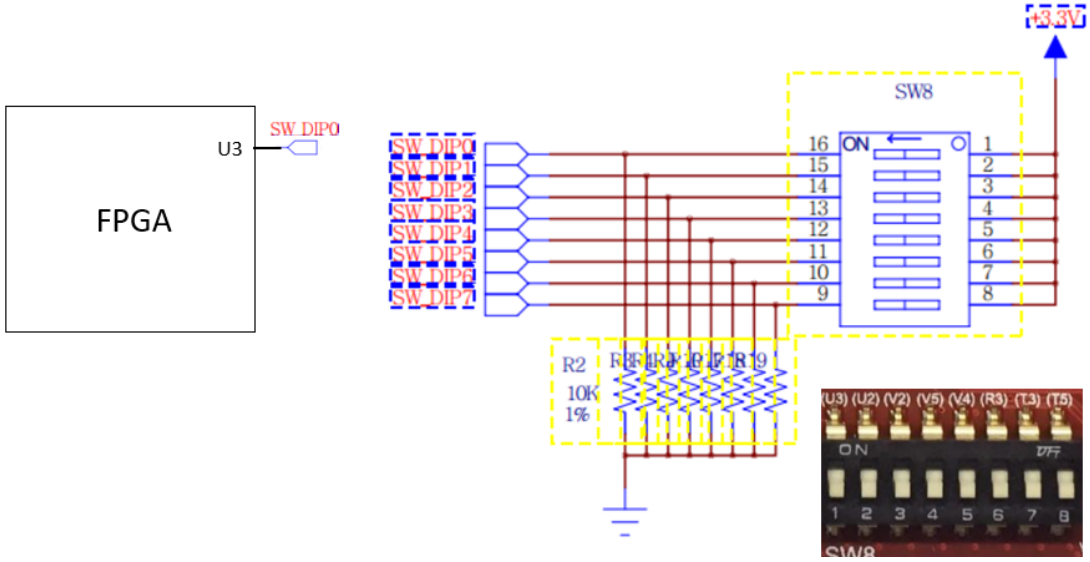


图 D.7: DIP 拨码开关原理图

管脚约束见手册。

D.5 | 七段数码管说明

EGO1 上共有 8 个七段数码管，DK1~DK8。

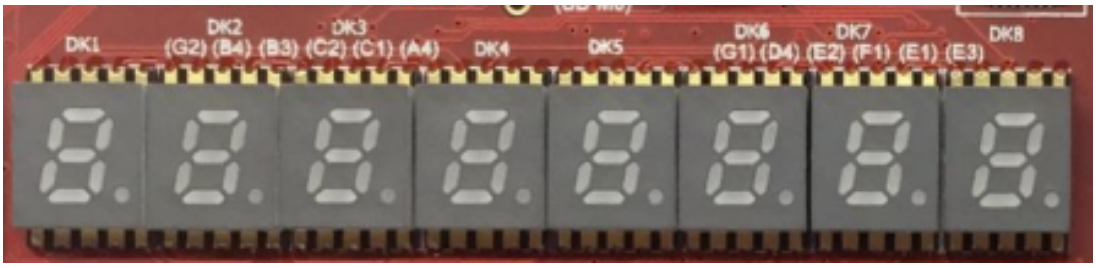


图 D.8: 七段数码管

如下图以 DK1 为例说明单个数码管的控制，每个数码管有 10 个管脚（A~G,DP,8，3)。

段选信号（高电平有效)：A~G 控制七段显示和 DP 控制小数点显示；

控制信号：8 和 3 位驱动管脚（信号线名称为 DN0_K1)。

其中, DK1 的 A~G 和 DP 分别通过信号线 LED0_CA~LED0_CG, LED0_DP, 连接到 FPGA 的 B4，A4，A3，B1，A1，B3，B2 和 D5 管脚。

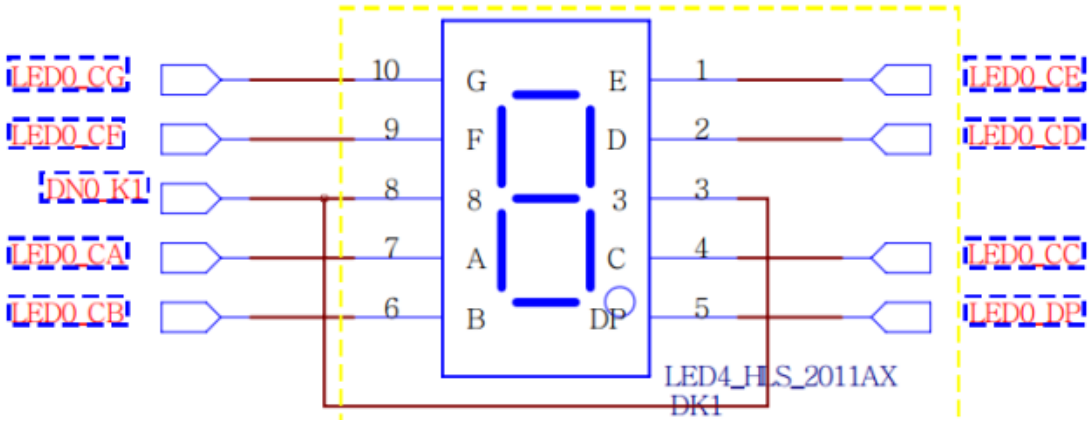


图 D.9: 七段数码管原理图

数码管的使能：数码管为共阴极数码管，即控制信号（DN0_K1）输入低电平使能数码管。如下图所示，控制信号由三极管驱动，通过信号线 LED_BIT1 连接到 FPGA 的 G2 管脚。因此，G2 高电平使能 DK1。

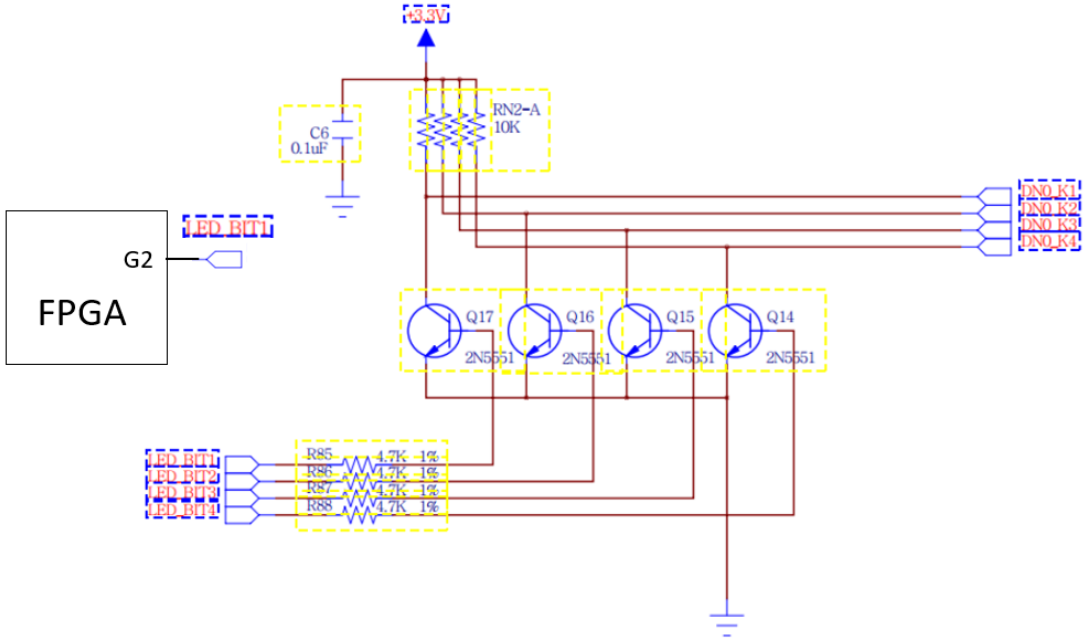
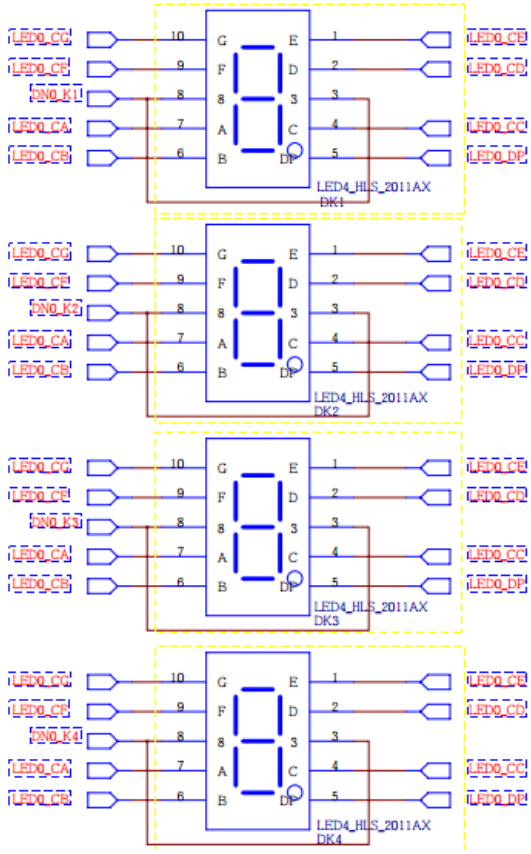


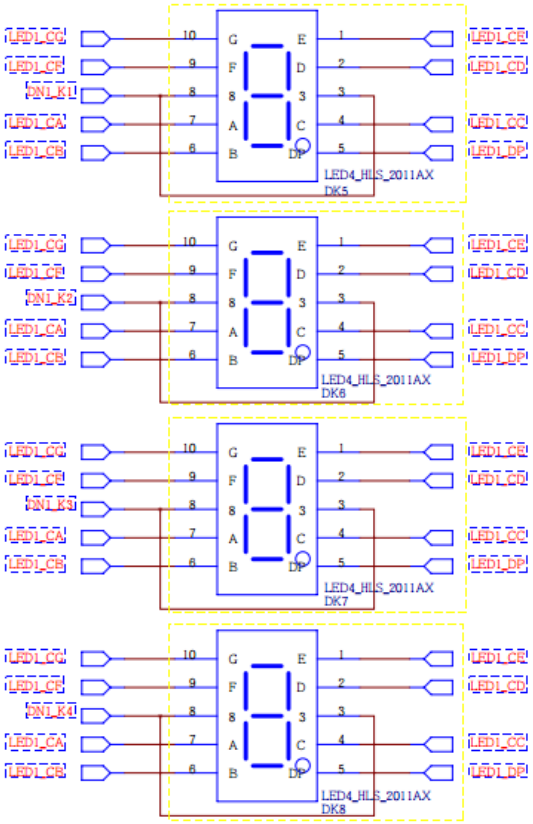
图 D.10: 数码管使能控制

数码管的分组管脚复用

如下面原理图所示，8 个数码管分为两组：



(a) 数码管原理图 1



(b) 数码管原理图 2

图 D.11: 数码管原理图

第一组：DK1∼DK4，如图可知所有 DK1∼DK4 的段选信号（A∼G,DP）管脚复用一组信号线（LED0_CA∼LED0_CG,LED0_DP）。例如：DK1∼DK4 的 A 管脚都连到了 LED0_CA。而 DK1∼DK4 的使能信号是单独控制的。如上图所示,DK1∼DK4 的使能管脚分别连到了 DN0_K1∼DN0_K4，链接到 FPGA 的使能控制信号分别是 LED_BIT1∼LED_BIT4。

第二组：DK5∼DK8，也是段选信号复用（LED1_CA∼LED1_CG,LED1_DP），使能信号单独控制 LED_BIT5∼LED_BIT8。

更详细的原理图参见 EGO1 电路原理图。

管脚约束见手册。