

# 实验 3 简单时序电路设计

## 1. 实验内容

实现数码管动态显示模块。

要求：

- 用分频器获得满足要求的时钟信号。
- 用 8 个数码管的动态显示模拟跑马灯效果。
- 用按键控制跑马灯停顿。
- 用存储器存放待显示数字。

## 2. 实验目的

1. 掌握 Verilog 语言的时序逻辑电路设计、实现、仿真、调试方法。
2. 掌握分频器、解码器和存储器等模块的设计与实现。
3. 学会如何进行模块参数的设置和传递。
4. 掌握用仿真测试程序 ( test bench ) 对模块进行测试和验证的方法。
5. 学习和巩固模块的层次设计方法。

## 3. 实验步骤

### 3.1 设计时钟分频器控制 led 灯闪亮

#### 3.1.1 实验内容：

- nexys4ddr 实验板的系统时钟信号由 E3 引脚提供，频率 100MHz，周期 10ns。设计一个频率为 1Hz（周期为 1s）的分频器 divider，可在模块中用 parameter 定义一个参数 delay，值为 50,000,000，同时定义一个变量 cnt 对系统时钟 clk 的上升沿进行计数。每当 cnt 计数到 delay（ $50,000,000 \times 10\text{ns} = 500,000,000\text{ns} = 0.5\text{s}$ ）时，将分频时钟信号 CK 的电平反转，即可得到周期为 1s 的分频时钟信号。在上层模块中实例化 divider 模块时，可传递新的参数来调整 delay 的值，得到合适频率的分频时钟信号。
- 设计上层模块 lab3\_1 实例化分频器模块 divider，用分频器模块的输出信号控制 led[0] 闪亮。
- 在上层模块中修改分频器模块的参数 delay，观察 led[0] 闪亮快慢的变化。
- 模块 divider 和 lab3\_1 定义的框架已经在下面给出，请按以上要求完善代码，并在实验板上演示效果。

### 3.1.2 实验步骤：

- 创建工程，添加分频器模块，分频后的时钟控制一个 LED 闪亮。

分频是指将时钟信号的频率进行降低的时序电路，例如：如果要将频率降低为原来的  $1/N$ ，就叫  $N$  分频。为便于高层模块改变分频系数  $N$ ，将  $N$  定义为参数。实例化时可以把参数传递进去：

```
divider #( 100000 ) D1 ( ... );
```

```

module divider(clk, clk_N);

input clk;                // 系统时钟

output clk_N;             // 分频后的时钟

parameter N = 100_000_000; // 1Hz 的时钟, N=fclk/fclk_N

reg [31:0] counter;       /* 计数器变量，通过计数实现分频。

                           当计数器从 0 计数到(N/2-1)时，

                           输出时钟翻转，计数器清零 */

always @(posedge clk) begin // 时钟上升沿

    ⋮                      // 功能实现

end

endmodule

```

- 添加约束文件：nexys4ddr 实验板的系统时钟信号由 E3 引脚提供，频率为 100MHz，将端口 clk 分配给 E3，clk\_N 分配给 LED0，使其由 clk\_N 控制 LED0 闪烁。

## 3.2 设计一个 3 位计数器

- 所谓计数器就是在时钟信号沿的控制下能够进行自动加或减（通常是+1 或-1）的时序电路。

```

module counter(clk, out);

input clk;                // 计数时钟

output [2:0] out;          // 计数值

always @(posedge clk) begin // 在时钟上升沿计数器加 1

    :                      // 功能实现

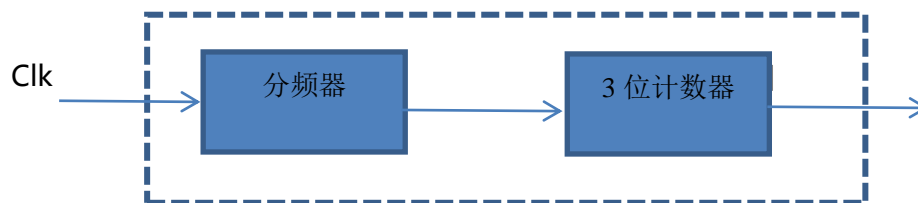
end

endmodule

```

### 3.3 计数器控制 LED 灯闪亮

- 添加一个包含分频器和计数器的上层模块。输入时钟分频后控制计数，输出计数值，用该计数值控制 3 个 LED。



- 修改约束文件：将端口 clk 分配给 E3，out[2:0]分配给 LED2~LED0。

### 3.4 设计只读存储器 ( ROM )

- 设计一个容量为 8 单元的 4 位只读存储器，里面存放待显示数字 0、2、4、6、8、A、C、E 的 4 位二进制编码，可在 initial 语句中对存储器进行初始化。

```
module rom8x4(addr, data);  
  
    input [2:0] addr;          // 地址  
  
    output [3:0] data;        // 地址 addr 处存储的数据  
  
    reg [3:0] mem [7:0];      // 8 个 4 位的存储器  
  
    initial    begin          // 初始化存储器  
        ....  
    end  
  
    ...                      // 读取 addr 单元的值输出  
  
endmodule
```

- 修改约束文件：用 3 个开关输入地址，输出的 4 位数据控制 4 个 LED。
- 将 rom8x4 设置为顶层文件（选中文件，右键，Set as top），再综合、实现、生成比特流。

### 3.5 设计 3-8 解码器和 7 段显示译码器

- 3-8 解码器 decoder3\_8 和 7 段显示译码器 pattern 已经在实验 2 中实现，在这里可

以直接使用，也可以按照下面的模块定义，将相应的代码直接复制过来。

```
module decoder3_8(num, sel);  
  
input [2: 0] num;          // 数码管编号：0~7  
  
output [7:0] sel;          // 7 段数码管片选信号，低电平有效  
  
┆                          // 功能实现  
  
endmodule
```

```
module pattern(code, patt);  
  
input [3: 0] code;          // 16 进制数字的 4 位二进制编码  
  
output [7:0] patt;          // 7 段数码管驱动，低电平有效  
  
┆                          // 功能实现  
  
endmodule
```

## 3.6 数码管动态显示

- 设计使数码管动态显示的顶层模块，实现八个数码管从右至左“分时”显示存储器中的数据，即 mem[0]值在 AN0 显示，mem[1]值在 AN1 显示，.....

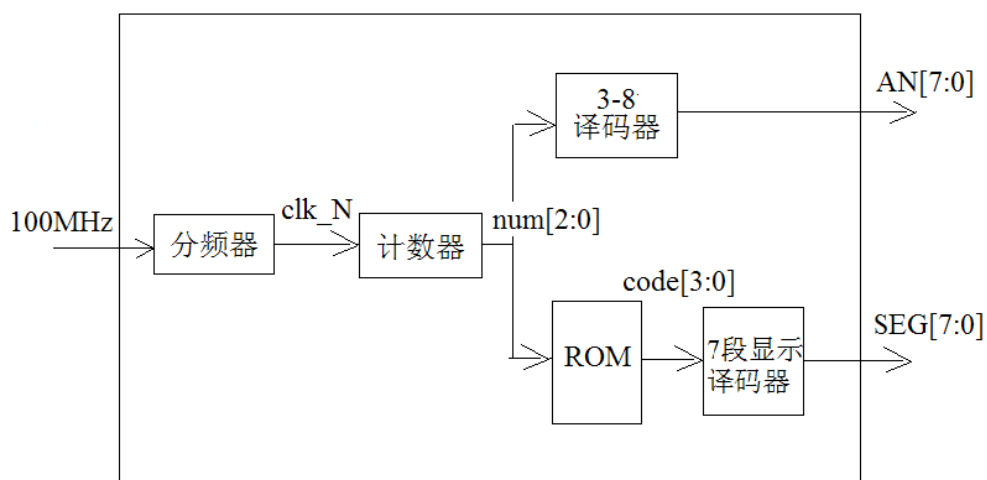
```
module dynamic_scan(clk, SEG, AN);  
  
    input clk;                // 系统时钟  
  
    output [7:0] SEG;          // 分别对应 CA、CB、CC、CD、CE、CF、CG 和 DP  
  
    output [7:0] AN;           // 8 位数码管片选信号
```

⋮

// 功能实现

endmodule

- 可以通过按照下图所示连接各个模块，采用结构建模的方法来设计实现 dynamic\_scan 顶层模块。



数码管动态扫描模块内部结构图

- 改变分频系数，观察数码管动态扫描的效果。人眼的视觉暂留时间大概在二十四分之一秒左右，所以，当扫描频率高于 24Hz 时就不会感觉到闪烁，视觉上每个数码管同时显示了不同数字。

### 3.7 进阶篇

- 增加三个开关，左中右；左右两个开关控制跑马灯的方向；中间开关控制停止跑动，八个数码管“同时”显示只读存储器中的数据。

