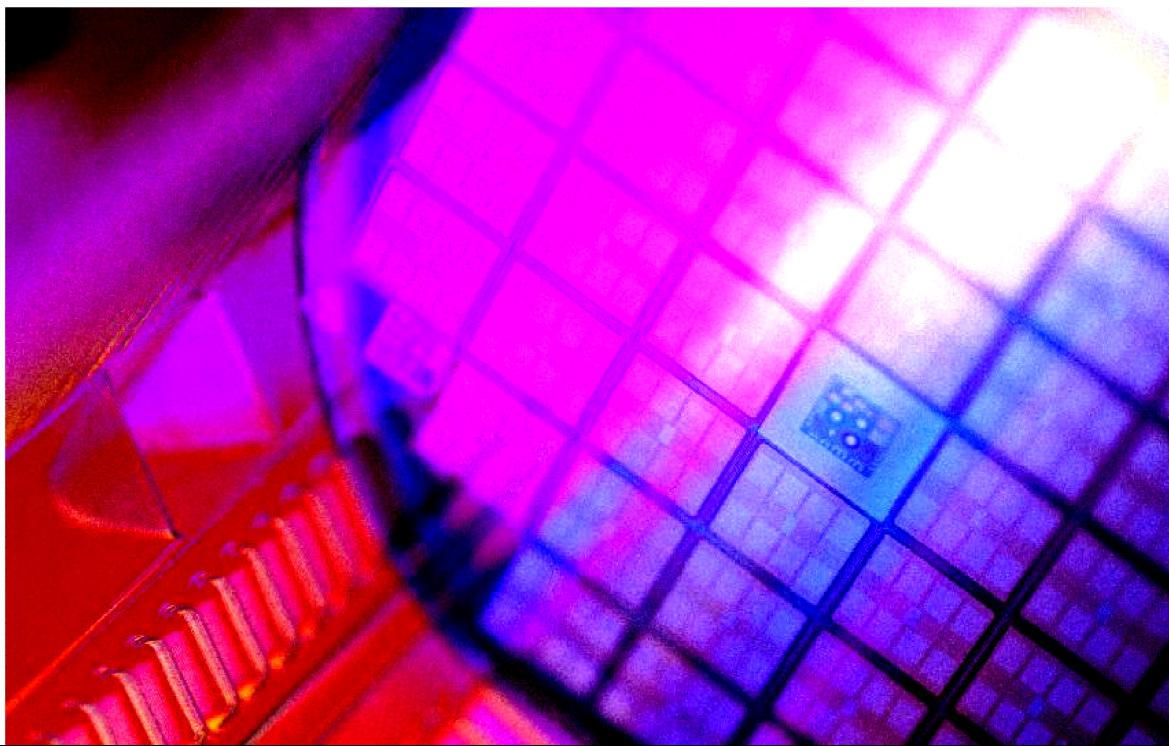


FEQ1301T01-V0.00



ZRtech

FPGA 开发套件 HDL 实验教程

—LED 实验



www.zr-tech.com

实验 1 LED 实验

1.1 第 1 例-闪灯

严肃的告诉各位观众 ,永远不要嫌弃闪灯 ;如同在 C 例子中不能嫌弃 Hello World 一样。

- C 中的 Hello World 例程至少可以告诉你以下信息 :

- 1、编译器已经正常安装 , 可以正常编译 ;
- 2、打印函数库是否准确调用 ;
- 3、输出选择是 Real UART 还是 JTAG UART 还是 IDE 终端。

后两者对于嵌入式开发调试中尤为重要。

- 同样的 , FPGA 的闪灯至少可以告诉你以下几点信息 :

- 1、FPGA 能否正常烧录 ;
- 2、烧录完成后的 FPGA 能否正常运行 ;
- 3、如果固化 Flash , 能否正常加载。

除此之外 , 闪灯还可以检测硬件好坏.....

本文不侧重软件使用及版本 , 重在 HDL 的运用和编程方法。

下面开闪 :

1.1.1 点亮 LED (example_led_1)

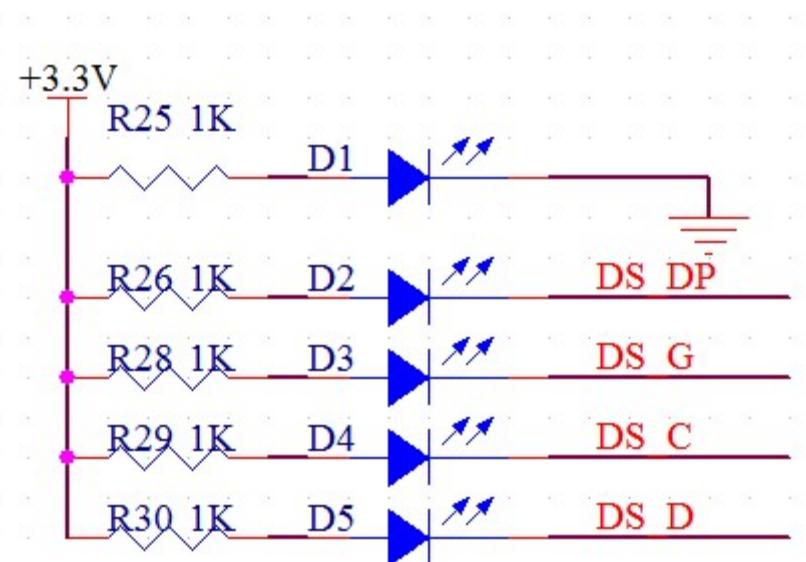


图 1 LED 原理图上的接法

在 Espier_I V2.00 开发板上由于资源丰富，因此所有 IO 都有分配，其中 LED 和数码管的管脚是复用的，因此信号是根据数码管命名的。

请看下面的代码，参见\ example_led_1：

```

1  module top(
2      //Dual Purpose Pin LED
3      DS_C, DS_D, DS_G, DS_DP
4  );
5
6  output DS_C, DS_D, DS_G, DS_DP;
7
8  assign DS_C = 1'b1;
9  assign DS_D = 1'b1;
10 assign DS_G = 1'b0;
11 assign DS_DP = 1'b0;
12
13
14 endmodule

```

其中 module 和 endmodule 配对出现，表示一段完整的代码。类似于 C 中的 “{” 和 “}” ，而 top 则是这段 module 的名字，类似于函数名。在 top 后的()中是端口列表(port list)，表示本段代码对外的输入和输出部分。

注意：端口用逗号 “,” 分割，但是最后一个端口是没有标点符号的，如果加上将有各种

意向不到的错误。

紧随其后的是端口的输入输出定义，不具体介绍了。

在新 Verilog-2001 规范中，端口和类型定义可以合在一起，如下，风格更加简洁。

```
1  module top(
2      //Dual Purpose Pin LED
3      output DS_C,DS_D,DS_G,DS_DP
4  );
5
6
7
```

这段代码是最简单的组合逻辑赋值，在 verilog 中最基本的 2 种变量就是线网型（ Wire ）和寄存器型（ Reg ），如果一个变量没有声明类型默认是 wire 型，但记住，类型是由变量使用环境决定的，不是定义出来的， wire 型变量代表了组合逻辑， reg 型代表了时序逻辑，如果你使用错误，综合器就会报告你的类型定义错误。

注意：verilog 中变量可以隐含线网型 wire ，但是很多编码规范和标准都要求将 wire 型显性的写出来，除了增强可读性之外也为增加代码的严谨度和规范性。

1.1.2 稍作改变（ example_led_2 ）

看完第 1 例，有可能有 2 个地方引起了你的注意：

- 1、变量的命名令人费解，这是因为信号命取自于复用的数码管，和 LED 压根对不上；
- 2、多希望用数组的方式来格式化一下资源名称，方便调用呢。

于是就有了下面的写法：

```
1  module top(
2    //Dual Purpose Pin LED
3    output DS_C,DS_D,DS_G,DS_DP
4  );
5
6  wire [3:0]led ;
7
8  assign {DS_C,DS_D,DS_G,DS_DP} = led;
9
10 assign led = 4'b0011;
11
12 endmodule
```

定义 1 个 4bit 的[3:0]led，将四个变量 DS_x 进行位拼接，所谓位拼接，类似于 C 里的结构体位段定义。

注意：4bit 的位宽定义是写在变量之前的，和 C 不同。如果你写在后面代表的是“寄存器组”，意义将会不同。

定义了变量 led 后，后面的赋值就清晰了很多，对 led 进行整体赋值，等效于之前的分开赋值。

这样书写更便于阅读，在逻辑只有几行的情况下，显得有些画蛇添足，但是当代码量逐渐增加的情况下，变量的定义清晰显得尤为重要。

1.1.3 标准闪灯 (example_led_3)

从点灯到闪灯是一个本质的变化，在这里引入了时序的概念，FPGA 中的 90%以上的功能都是由时序设计完成的。在基本组成元素中非时序组合逻辑在 FPGA 内部使用查找表资源实现(Look-up Table , LUT)。而时序则由寄存器实现。(如果暂时不理解，这段可以视而不见)

```

1  module top(
2      //Clock Input:48M
3      input CLK,
4      //Dual Purpose Pin LED
5      output DS_C,DS_D,DS_G,DS_DP
6  );
7
8      //定义一个参数为下面以秒计数做准备
9      parameter SEC_TIME = 48_000_000;
10
11     wire [3:0]led ;
12     assign {DS_C,DS_D,DS_G,DS_DP} = led;
13
14     //定义计数器，并初始化为0
15     //此处初始化仅对仿真有效，综合器会自动无视，下同
16     reg [31:0] cnt1;
17     initial cnt1 = 32'b0;
18     //定义hz级时钟
19     reg clk_hz;
20     initial clk_hz = 1'b0;
21
22     //标准计数器一只
23     always@(posedge CLK)
24         if(cnt1 == SEC_TIME/2)
25             begin
26                 cnt1 <= 0;
27                 clk_hz = !clk_hz;
28             end
29         else cnt1 <= cnt1 + 1;
30
31     assign led = {clk_hz,clk_hz,clk_hz,clk_hz};
32
33 endmodule

```

计数器是 Module 中最常用的“零部件”之一，计数器有很多种形式，这个例子中使用的是“计到固定值复位”模式。这名字是自己取得，实质上在硬件上代表着计数器输出进入比较器和固定数值相比，结果反馈至计数器的复位端。

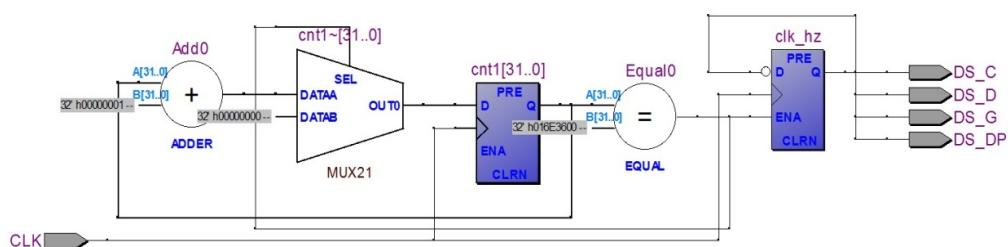


图 2 计数器的 RTL 图

综合器：将 HDL 语言翻译成 RTL 网表的工具，是非常之关键的一环。

图中是 Quartus 自带的综合器生成的综合图。不同的综合器生成的综合图可能是不一样的，综合器的优劣在相当程度上决定了设计实现的好坏，不多说了，你会懂的。

1.1.4 计数闪灯 (example_led_4)

在 CPU 中使用定时器完成的任务，对于并发的 FPGA 来说完全是小菜一碟，因为 32bit 的计数器只要资源允许可以随意构造，且并发工作，计数闪灯因此非常简单，只需要在闪灯的基础上将最后一句赋值改为：

```
assign led = {cnt1[24],cnt1[23],cnt1[22],cnt1[21]};
```

也就是将计数器的 4 个位直接赋值给了 LED，实现的就是计数灯的效果。计数器的值在 RTL 视图中被隐藏了，因此看起来有点费解。

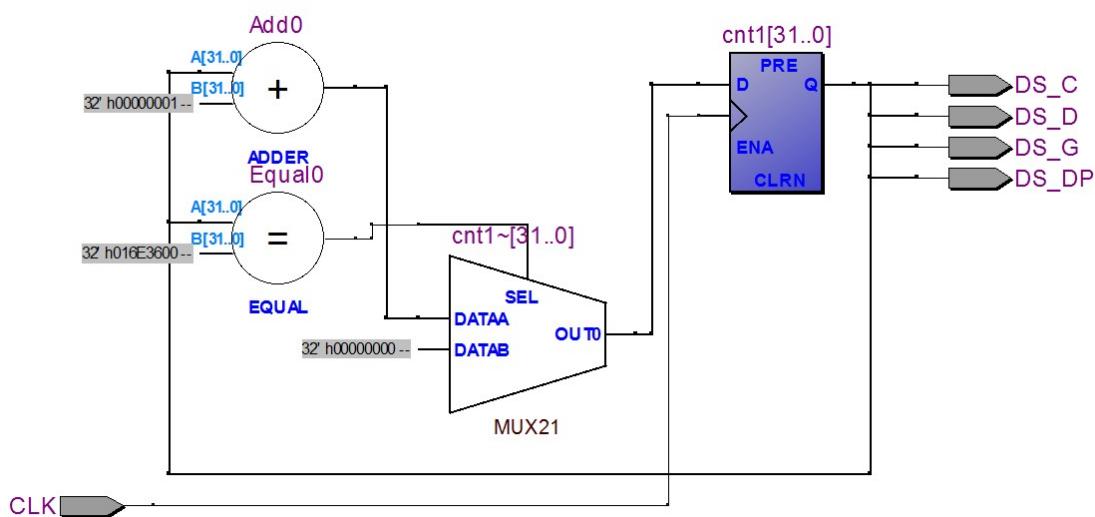


图 3 计数闪灯的 RTL 图

1.1.5 移位闪灯 (example_led_5)

这就是俗称的“跑马灯”，一个灯闪过来闪过去，或者一溜灯闪过来闪过去。在 FPGA 中就是使用移位寄存器的方式来实现的。

- 方式 1 逐个变亮：

```
23 //标准计数器一只
24     always@(posedge CLK)
25         if(cnt1 == SEC_TIME/2)
26             begin
27                 cnt1 <= 32'b0;
28                 clk_hz = !clk_hz;
29             end
30         else cnt1 <= cnt1 + 1'b1;
31 //移位寄存器
32 reg [3:0]led_reg;
33 always@(posedge clk_hz)
34 begin
35     if(led_reg == 4'b1111)
36         led_reg = 4'b0000;
37     else
38         led_reg = {led_reg[2:0],1'b1};
39     end
40
41 assign led = led_reg;
42
43 endmodule
```

- 方式 2 逐个变暗：

```
42 //移位寄存器 逐个变暗
43 reg [3:0]led_reg;
44 always@(posedge clk_hz)
45 begin
46     if(led_reg == 4'b1111)
47         led_reg = 4'b0000;
48     else
49         led_reg = {led_reg[2:0],1'b1};
50     end
```

- 方式 3 单灯遍历：

```
52 //移位寄存器 单灯遍历
53 reg [3:0]led_reg;
54 always@(posedge clk_hz)
55 begin
56     if(led_reg == 4'b0000)
57         led_reg = 4'b1110;
58     else if(led_reg == 4'b1111)
59         led_reg = 4'b1110;
60     else if(led_reg == 4'b1110)
61         led_reg = 4'b1101;
62     else if(led_reg == 4'b1101)
63         led_reg = 4'b1011;
64     else if(led_reg == 4'b1011)
65         led_reg = 4'b0111;
66     else if(led_reg == 4'b0111)
67         led_reg = 4'b0110;
68     else if(led_reg == 4'b0110)
69         led_reg = 4'b1110;
70     else
71         led_reg = 4'b1110;
72     end
73
```

- 方式 4 双灯遍历：

```

74 //移位寄存器 双灯遍历
75 reg [3:0]led_reg;
76 always@(posedge clk_hz)
77 begin
78     if(led_reg == 4'b0000)
79         led_reg = 4'b1100;
80     else if(led_reg == 4'b1111)
81         led_reg = 4'b1100;
82     else if(led_reg == 4'b1100)
83         led_reg = 4'b1001;
84     else if(led_reg == 4'b1001)
85         led_reg = 4'b0011;
86     else if(led_reg == 4'b0011)
87         led_reg = 4'b0110;
88     else if(led_reg == 4'b0110)
89         led_reg = 4'b1100;
90     else
91         led_reg = 4'b1100;
92 end

```

请注意后面两种方式的写法，这实际上是一种万能的写法，移位寄存器如果使用这种方式

可以写的十分通俗易懂，而有别于前面的位拼接方式的写法。这种写法实质上就是状态机。

请看 RTL 的结果：

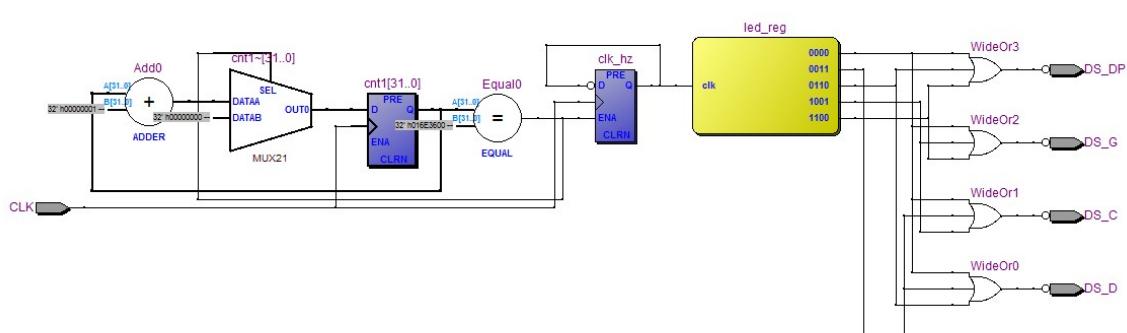


图 4 双灯遍历的 RTL 图

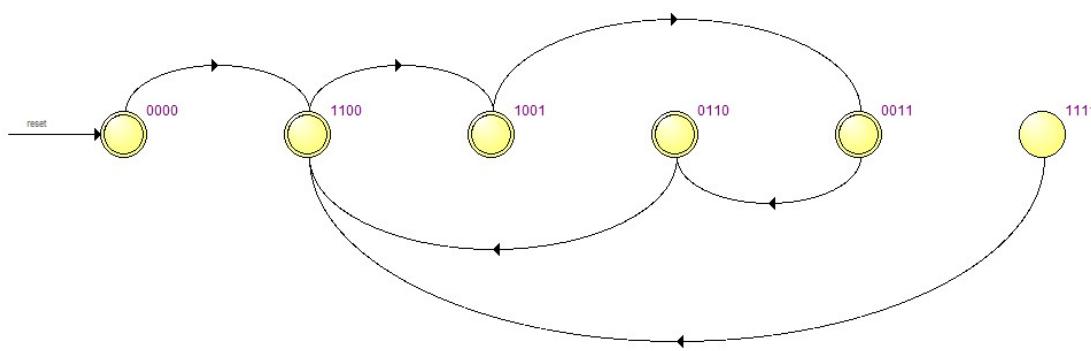


图 5 双灯遍历的状态跳转图

状态机有多种形式的写法，这种虽然不是最标准的但是最容易理解和入门的一种。记住在 HDL 中不管你 how 写，主要不写错代码，最终 RTL 综合的结果都是一样的。
所以，当你开始抱怨代码不灵时请先看一眼 RTL 图，你会发现很多问题，譬如：你以为写了一个状态机，但是 RTL 综合却无法提取状态转换，那必须是你写法上的问题。

利用这种写法你可以写出任意的变化结构，注意开始的 2 种状态，那是一种保护，当状态机进入初始或者跳转到未知状态时必须让其跳转到正常状态中，最后 else 也是这个目的。

1.1.6 闪灯总结

闪灯这章结束了，这章说的是基本的用法和例子，位拼接、计数器、状态机等都是最常用的“部件”之一。HDL 写法千变万化，但 RTL 只有你期望的那种是正确的。这儿告诉你不要用 C 语法去理解 Verilog，虽然他们的确很像。不论你在写什么，一定要知道 RTL 生成了什么。

说的有点抽象，举一个例子。现在的综合器都有物理综合的功能，说穿了就是根据 FPGA 内部结构来决定综合的结构，譬如现在 FPGA 中都有 DSP 单元，实际就是乘累加的结构，当你写出一个 $a=b+c$ 时，综合器会根据 b 和 c 是否常数，分别的位宽，甚至运行的速度，来决定是否启用 DSP 单元，一般有一个默认的阈值，比如位宽上达到多少位后才会启用，否则只会使用基本的 LUT 和寄存器资源搭建而成乘和加，当然也可以通过手工的约束来通知综合器

希望做成什么样的结构，最后还可以使用 IP 来跳过综合器，直接使用芯片内部资源……手段很多，闪灯就说这么多了。

文档内部编号 : FEQ1301T01

编号说明 :

首一字母 : F-FPGA 系列

首二字母 : L-理论类 E-实验类 T-专题类

首三字母 : C-普及类 Q-逻辑类 S-软核类

数字前两位 : 代表年度

数字后两位 : 同类文档顺序编号

尾字母/数字 : C 目录 , T 正文 , 数字表示章节号

修订记录

版本号	日期	描述	修改人
0.00	2013.9.25	FEQ1301T01 文档建立	kdy