CHAPTER 3 第3章

组合逻辑电路设计与应用

本章重点介绍以下组合逻辑电路的设计：基本门电路设计、比较器、数据选择器、七段译码器、编码器、译码器、ALU 等电路；同时，结合上述项目介绍了 FPGA 设计中常用的一些工具，包括：ISim、FPGA Editor、PlanAhead、ISE schematic viewer、Design Summary 等。

学习组合逻辑电路的设计与应用，主要目标有 3 个：①通过实践掌握 ISE 这个工具软件的使用方法；②通过实践进一步掌握 HDL 语言结构；③通过实践掌握 ISE 工具软件的使用方法。

3.1 基本门电路

1. 设计要求

设计完成两输入与门、与非门、或门、或非门、异或门、同或门。

要求：

(1) 将拨码开关 SW0 和 SW1 分别作为输入变量 a 和 b；y 输出接到 6 个灯 LD0～LD5，通过拨动拨码开关来观察输出的状态变化。

(2) 学习使用 ISim 进行功能仿真的方法和技巧。

2. 基本门电路设计

本例为组合逻辑，使用数据流描述来实现，如例 3-1 所示。

【例 3-1】 二输入逻辑门实现代码。

```
module P1_gate2(a,b,y);
input a;
input b;
output[5:0] y;
assign y[0]=a&b;            //and
assign y[1]=~(a&b);         //nand
assign y[2]=a|b;            //or
assign y[3]=~(a|b);         //nor
assign y[4]=a^b;            //xor
assign y[5]=~(a^b);         //xnor
endmodule
```

为了引脚锁定的统一和方便，下面给例 3-1 增加顶层模块。

【例 3-2】 对例 3-1 增加顶层模块。

```
module P1_gate2_top(bmkg,led);
input[1:0] bmkg;
output[5:0] led;
P1_gate2 U1(.a(bmkg[0]),
            .b(bmkg[1]),
            .y(led));
endmodule
```

对例 3-2 增加引脚锁定,约束文件如例 3-3 所示。

【例 3-3】 对例 3-2 设计的管脚约束文件。

```
NET "bmkg[0]" LOC=P11;        //bmkg0
NET "bmkg[1]" LOC=L3;         //bmkg1
NET "led[0]" LOC=M5;          //LD0
NET "led[1]" LOC=M11;         //LD1
NET "led[2]" LOC=P7;          //LD2
NET "led[3]" LOC=P6;          //LD3
NET "led[4]" LOC=N5;          //LD4
NET "led[5]" LOC=N4;          //LD5
```

最后对已经约束管脚的设计进行综合、实现、生成配置文件、编程到 Basys2 开发板。在 Basys2 开发板上,拨动 SW0、SW1 这两个拨码开关,就可以看到 LD0～LD5 这 6 个 LED 灯会相应变化,并指示当前的输出状态。

在使用 Basys2 开发板时,需要为输出/输入信号指定管脚,因此我们将 Basys2 开发板所有输入和输出引脚都整理在 basys2.ucf 文件中,如例 3-4 所示。

【例 3-4】 basys2.ucf 文件。

```
#pin assignment for clock
NET "clk" LOC=B8;             //MCLK
#pin assignment for slide switches
NET "bmkg[0]" LOC=P11;        //SW0
NET "bmkg[1]" LOC=L3;         //SW1
NET "bmkg[2]" LOC=K3;         //SW2
NET "bmkg[3]" LOC=B4;         //SW3
NET "bmkg[4]" LOC=G3;         //SW4
NET "bmkg[5]" LOC=F3;         //SW5
NET "bmkg[6]" LOC=E2;         //SW6
NET "bmkg[7]" LOC=N3;         //SW7
#pin assignment for leds
NET "led[0]" LOC=M5;          //LD0
NET "led[1]" LOC=M11;         //LD1
NET "led[2]" LOC=P7;          //LD2
NET "led[3]" LOC=P6;          //LD3
NET "led[4]" LOC=N5;          //LD4
NET "led[5]" LOC=N4;          //LD5
NET "led[6]" LOC=P4;          //LD6
NET "led[7]" LOC=G1;          //LD7
```

```
#pin assignment for 7-segment displays
NET "duan[0]" LOC=L14;          //CA
NET "duan[1]" LOC=H12;          //CB
NET "duan[2]" LOC=N14;          //CC
NET "duan[3]" LOC=N11;          //CD
NET "duan[4]" LOC=P12;          //CE
NET "duan[5]" LOC=L13;          //CF
NET "duan[6]" LOC=M12;          //CG
NET "duan[7]" LOC=N13;          //DP
NET "wei[0]" LOC=F12;           //AN0
NET "wei[1]" LOC=J12;           //AN1
NET "wei[2]" LOC=M13;           //AN2
NET "wei[3]" LOC=K14;           //AN3
#pin assignment for pushbotton switches
NET "key[0]" LOC=G12;           //BTN0
NET "key[1]" LOC=C11;           //BTN1
NET "key[2]" LOC=M4;            //BTN2
NET "key[3]" LOC=A7;            //BTN3
#pin PS2 interface
NET "PS2C" LOC=B1;              //PS2C
NET "PS2D" LOC=C3;              //PS2D
#pin VGA interface
NET "R[0]" LOC=C14;             //red0
NET "R[1]" LOC=D13;             //red1
NET "R[2]" LOC=F13;             //red2
NET "G[0]" LOC=G14;             //green0
NET "G[1]" LOC=G13;             //green1
NET "G[2]" LOC=F14;             //green2
NET "B[0]" LOC=J13;             //blue0
NET "B[1]" LOC=H13;             //blue1
NET "HS" LOC=J14;               //hs
NET "VS" LOC=K13;               //vs
```

在例 3-4 的 basys2.ucf 文件中，在每个管脚指定行的最后都有一个注释，该注释对应着该管脚在开发板上的资源名称。basys2.ucf 文件涵盖了开发板上所有的可用资源，但在实际应用中，可能只会用到其中的一部分资源，此时用到的资源就指定相应的管脚，未用到的资源就不需要指定管脚。如例 3-2 仅用到了两个拨码开关和 6 个 LED 灯，所以进行引脚锁定时仅锁定这些资源，如例 3-3 所示。

对于本设计，可以不用例 3-2，直接对例 3-1 进行指定管脚，然后进行综合、实现、生成配置文件、编程到 Basys2 开发板。使用例 3-2 的好处在于，一是规范设计中与 FPGA 的管脚连接的信号的名称；二是可以方便地使用例 3-4 所示的约束文件。后续的设计中通常也会增加类似的约束文件。

3. 使用 ISim 进行功能仿真

关于 ISim 的使用方法，在前面“基于 ISE 的开发流程”一节已作了简单的说明。

可以对例 3-1 进行功能仿真，通过改变输入信号来观察输出信号的变化。例 3-1 的

测试激励如例 3-5 所示。

【例 3-5】 例 3-1 的 testbench。

```
module P1_gate2_test;
    //Inputs
    reg a;
    reg b;
    //Outputs
    wire [5:0] y;
    //Instantiate the Unit Under Test (UUT)
    P1_gate2 uut (
        .a(a),
        .b(b),
        .y(y)
    );
    //test simulus
    initial begin
        repeat(2) begin
            a=0;
            b=0;
            #100;
            a=0;
            b=1;
            #100;
            a=1;
            b=0;
            #100;
            a=1;
            b=1;
            #100;
        end
    end
endmodule
```

运行例 3-5，可得到仿真波形如图 3-1 所示。

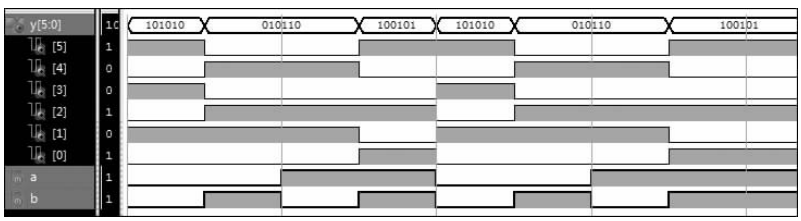


图 3-1 例 3-5 对应的仿真波形图

从仿真波形图，可以看出输出按照例 3-1 中所设定的逻辑随着输入的变化而变化。

下面介绍 ISim 仿真软件的一些常用的、非常实用的功能。

(1) 更改数据显示格式

ISim 在仿真时默认是二进制格式，为了便于使用，可以更改其显示的格式，右击需要

更改显示格式的数据，在弹出的快捷菜单上选择 Radix 命令，弹出可以选用的显示格式，如图 3-2 所示，包括：Binary(二进制)、Hexadecimal(十六进制)、Unsigned Decimal(无符号十进制数)、Signed Decimal(有符号十进制数)、Octal(八进制)、ASCII(ASCII 码)。

如果发现数据高低位反向，可以选择 Reverse Bit Order(反转 bit 顺序，即高位和地位对换)命令。

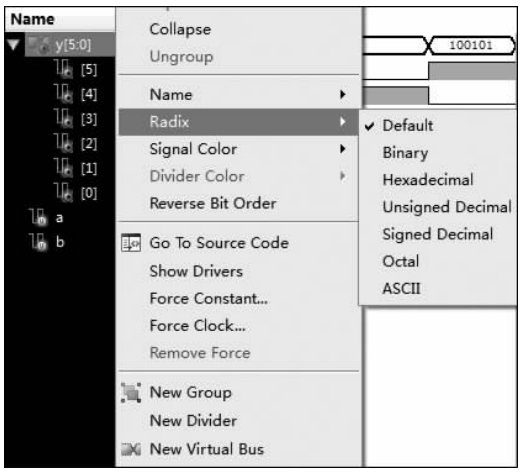


图 3-2　更改显示格式和位序

对于 1bit 数据，是一条线，如图 3-3 中变量 a 的波形所示。也可以选中这个数据右击，选择 New Virtual Bus 命令，然后修改名字为原来的信号名字，这样就将一条线变成了虚拟的总线形式，如图 3-3 中的变量 b 所示。

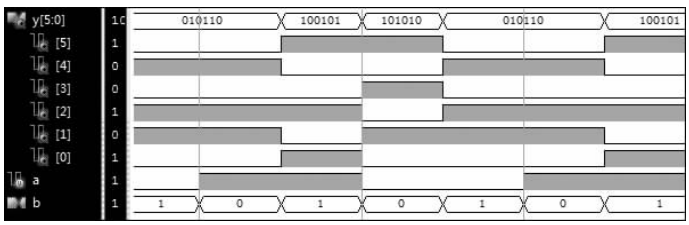


图 3-3　虚拟总线示例

(2) 查看中间变量

模块与模块之间的某些信号的变化，或者模块内部的某些信号的变化，有时是特别重要的，尤其是状态机的运行，我们在代码调试时经常会用到这些中间变量。

在 Instances and Processes Name 窗口中可选择层次设计中的任意模块，在右侧窗口 Object Name 中会出现此模块内的各种内部信号。此时可选择需要添加到波形文件的信号名字，然后直接拖到波形文件列表中，也可以右键添加至波形文件列表中，或者按 Ctrl＋W 组合键添加对应信号到波形文件中。

(3) 断点调试

断点调试是一个十分方便、有用的功能，可以查看指定位置是否有错误，方便 debug

程序。因为，HDL 代码是并行执行的，更多的时候是查看波形是否正确，通过波形发现错误，进而定位到对应的语句或者状态，然后修正错误。

在 Instances and Processes Name 窗口中，双击对应模块，则可打开对应的.v 文件，然后在需要的地方单击按钮(F9)加入断点。单击 run all 按钮(F5)运行，即可运行到断点处，然后可以单击单步 step(F11)执行按钮，查看中间运行结果。

(4) 查看 Memory

很多时候，需要查看设计的存储空间是否正确的存储了所需的值。设计中用到的存储空间包括 ROM、RAM、寄存器堆等，这些存储空间可能是通过 HDL 写的，也可能是通过 IP 核使用分布式 RAM 或块 RAM 构建的。

单击 Memory 窗口，然后双击需要显示的内存空间，则可打开对应的 Memory 空间。如果没有发现 Memory 窗口，可以选择菜单栏的 View|Panels 命令。

默认显示的数值为二进制，可以修改数据显示的格式(二进制、十进制有符号数、十进制无符号数、十六进制、八进制、ASCII 码)，同时也可以修改地址显示的格式。如果要查找某个地址的值，只需在地址栏中输入这个值，按回车键即可。

(5) 测量时间

在有些时候，需要测量两个信号之间的时间间隔。如果只是简单地测量两个边沿的时间间隔，如图 3-4 所示，可先按着鼠标左键选中一个边沿，然后拖动鼠标到另一个边沿，此时在波形的下面将出现时间轴，则可测量两个上升沿之间的时间。

如果需要测量的时间太多，可以添加 Marker，单击要加入标记的地方，然后单击标记按钮，或者右击在弹出的快捷菜单中选择 Markers|Add Marker 命令，此时并不能出现时间轴，单击 Marker 线，蓝色的线将变成白色，并以此为时刻 0 点，此时就可以看到时间间隔，如图 3-5 所示。

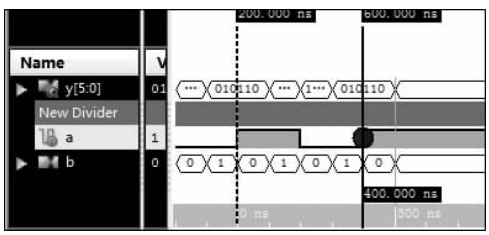


图 3-4 添加分组、测量 a 两个沿的时间间隔

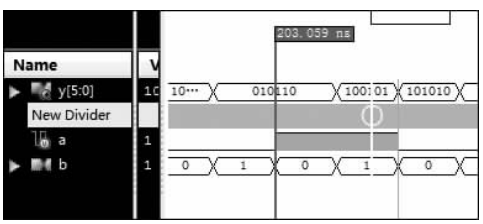


图 3-5 添加 Marker

(6) 创建分组和添加分割块

添加分割块是添加一个实心方块，可以将不同模块之间的信号分隔开，方便查看。在波形文件的 Name 栏的空白处右击，弹出图 3-2 所示的菜单。其中的 New Divider 命令用于分割，New Group 命令用于分组。使用 New Group 命令时，首先要选中需要加入分组的信号，然后右击，在弹出的快捷菜单中选择 New Group 命令，就可以修改相应的用于分组的名字。

(7) 保存仿真信息

为了方便再次仿真时，能够保存此次仿真的所有设置，可以保存波形文件，选择 File|Save As 命令输入文件名字，此后对该文件可以继续操作，在关闭 ISim 前记得要保存波

形文件。再次仿真时，ISim 不会直接调用以前保存的波形文件，而是一个 defalut. wcfg，此时只需通过选择 File|Open 命令打开以前保存的波形文件，然后就可以在以前设置的基础上继续仿真了。

4. 拓展练习

在互联网上或通过其他途径查找以下芯片的功能，然后使用 FPGA 予以实现，并在 Basys2 开发板上予以验证。

(1) 4 双输入与非门 74LS00。

(2) 4 异或门 74LS86。

(3) 4 双输入与非门 74LS20。

(4) 4-2-3-2 输入与或非门 74LS64。

3.2　比较器电路

1. 设计要求

实现两个 2 位数的比较器，并在开发板上验证。

要求：

(1) 拨码开关 SW0、SW1 作为输入变量 a；拨码开关 SW2、SW3 作为输入变量 b；y 输出接到了 3 个灯 LD0～LD2。如果 a 大于 b，LD0 亮；如果 a 小于 b，LD2 亮；如果 a 和 b 相等，则 LD1 亮。

(2) 学习 FPGA Editor 察看实现细节，查看 FPGA 资源利用情况的方法。

2. 比较器设计

两个 2 位数的比较器可以使用行为语句建模，其实现代码如例 3-6 所示。

【例 3-6】 两个 2 位数的比较器的代码实现。

```
//顶层模块
module  P2_compare_top(bmkg,led);
input[3:0] bmkg;
output[2:0] led;
P2_compare2 U1(.a(bmkg[1:0]),
               .b(bmkg[3:2]),
               .y(led));
endmodule
//设计实现
module P2_compare2(a,b,y);
input[1:0] a;
input[1:0] b;
output reg[2:0] y;
always @(a,b)
    if(a>b) y=3'b001;
    else if(a<b) y=3'b100;
    else y=3'b010;
endmodule
```

结合例 3-4 对本设计的输入和输出指定管脚，然后进行综合、实现、生成配置文件、编程到 Basys2 开发板。通过拨动 SW0～SW3 这 4 个拨码开关，观察 LD0～LD2 这 3 个 LED 的输出状态。

3. 使用 FPGA Editor 察看 FPGA 实现细节

利用 FPGA Editor，可以察看 FPGA 器件内部的结构，可以查看我们的设计用到了哪些 FPGA 资源，以及在 FPGA 内部的实现细节，还可以动调整设计所用的 FPGA 资源，并修改在 FPGA 内部实现细节。

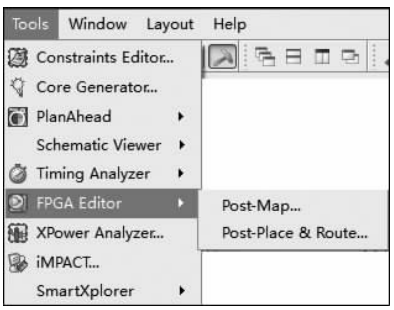


图 3-6　FPGA Editor 打开路径

选择 Tools|FPGA Editor 命令，如图 3-6 所示。

在图 3-6 中，FPGA Editor 工具里面有两个选项，Post-Map...选项是手动布局布线时使用的工具，而 Post-Place & Route...选项则是察看布局布线结果的工具。本节仅使用 Post-Place & Route...工具来察看实现的结果，关于使用 Post-Map...选项进行手动布局布线，本书不作介绍，感兴趣的读者可以查阅相关资料来学习使用。

首先，我们选择 Post-Place & Route...命令察看例 3-6 实现后的布局布线结果，如图 3-7 所示。

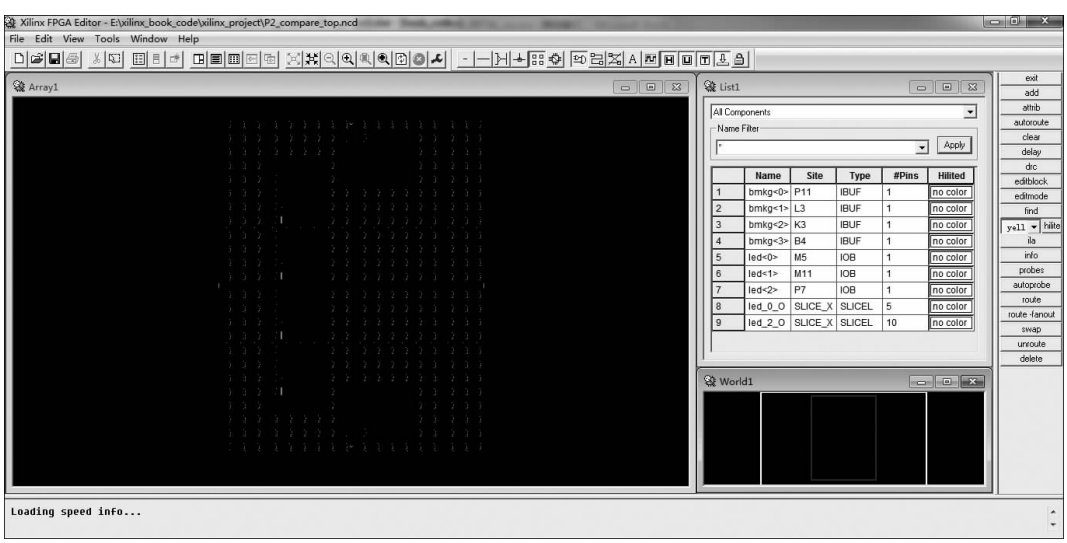


图 3-7　选择 Post-Place & Route...命令出现的界面

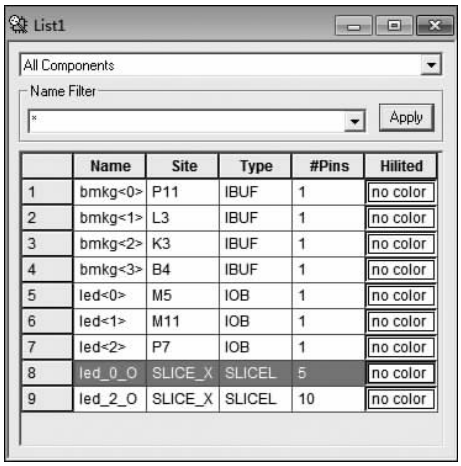
FPGA Editor 有 4 个主要窗口：列表(List)、全局(World)、阵列(Array)和块(Block)。列表(List)窗口显示设计中所有活动的项目。通过此窗口顶部的下拉菜单可选择其内容，列表内容包括已经布局或还未使用的部件、网络或未布线的网络等。全局视图(World)窗口始终显示完整 FPGA 硅片视图，这在试图确定某个网络的布线情况时非常有用。同时，阵列(Array)窗口则是 FPGA 构造和逻辑的动态视图。如果双击

Array 视图中的任何项目，会显示 Block 视图，给出所选择项目或逻辑单元的详细情况。

在 Array 视图和 Block 视图，可以按住 Ctrl 键和 Shift 键，然后利用鼠标滚轮对视图进行放大/缩小、上移/下移，操作起来非常便捷。

图 3-7 中的 Ist 窗口，放大后如图 3-8 所示。

双击图 3-8 中的第 8 项 led_0_O，弹出图 3-9，从图中可以看出实现 led[0]输出所用的资源以及资源的位置信息，资源在 Array 窗中用红色块标识，资源所处的位置在 Word 窗口中示意。如果使用 Post-Map...选项工具，就可以手动选择实现 led[0]的资源。



	Name	Site	Type	#Pins	Hilited
1	bmkg<0>	P11	IBUF	1	no color
2	bmkg<1>	L3	IBUF	1	no color
3	bmkg<2>	K3	IBUF	1	no color
4	bmkg<3>	B4	IBUF	1	no color
5	led<0>	M5	IOB	1	no color
6	led<1>	M11	IOB	1	no color
7	led<2>	P7	IOB	1	no color
8	led_0_O	SLICE_X	SLICEL	5	no color
9	led_2_O	SLICE_X	SLICEL	10	no color

图 3-8　List 窗口

双击图 3-9 中的红色块，打开实现 led[0]的 Block，如图 3-10 所示，可以看到 led[0]实现的细节信息。

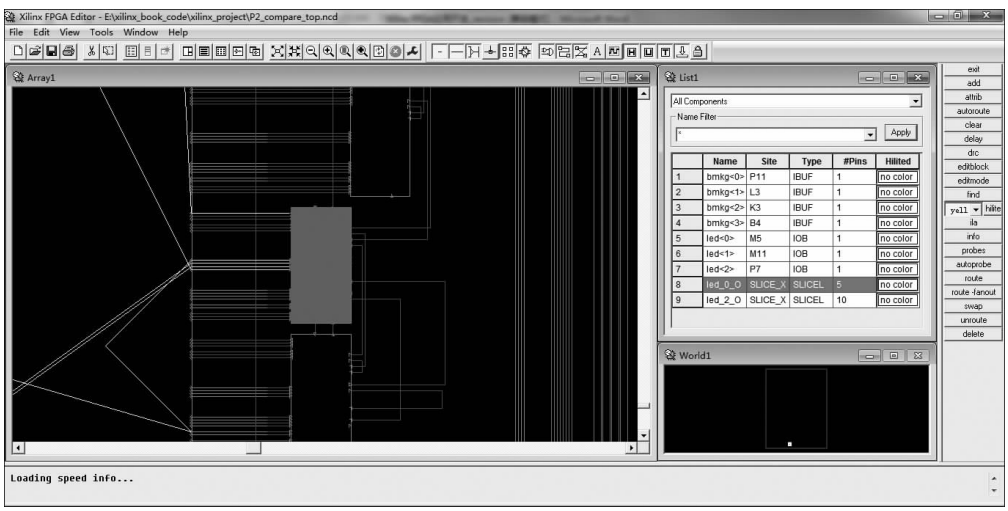


图 3-9　led[0]实现所用的资源以及资源的位置信息

从图 3-10 中可以看到 led[0]实现时用到了一个 LUT 和一个 MUX。单击图 3-10 上方的 F=按钮，会显示 led[0]的实现方程，如图 3-11 所示。如果使用 Post-Map...命令，就可以手动修改该实现方程，从而改变 led[0]的功能，此时不用修改 RTL 代码就可以直接在此基础上实现，显然这对于快速修改实现细节和快速验证无疑很有帮助。

根据上面的介绍，我们可以看到 FPGA Editor 强大的功能。利用 FPGA Editor，不仅可以察看 FPGA 器件内部的结构，而且可以查看我们的设计用到了哪些 FPGA 资源，以及在 FPGA 内部的实现细节，还可以动调整设计所用的 FPGA 资源，并修改在 FPGA 内部实现细节。对 FPGA 内部结构感兴趣的读者，可以参考介绍 FPGA 结构的书籍并结合 FPGA Editor 来理解学习，一定会有事半功倍的效果。

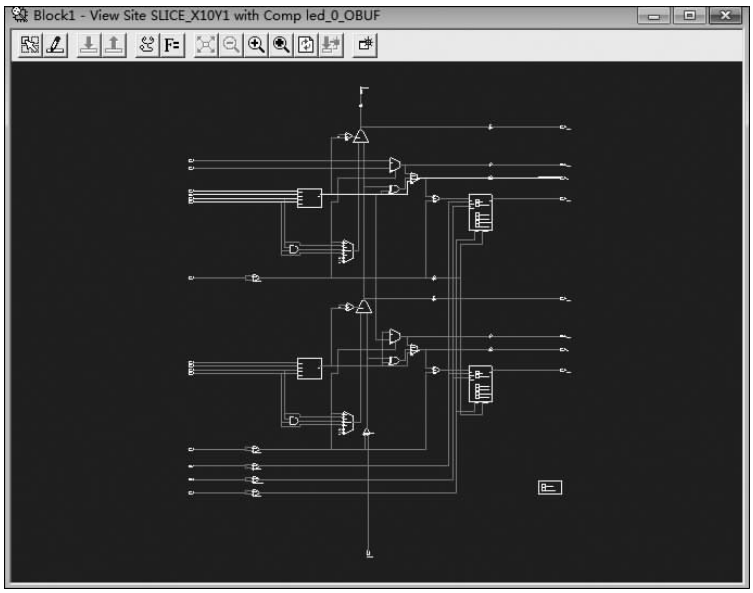


图 3-10　led[0]实现的细节信息

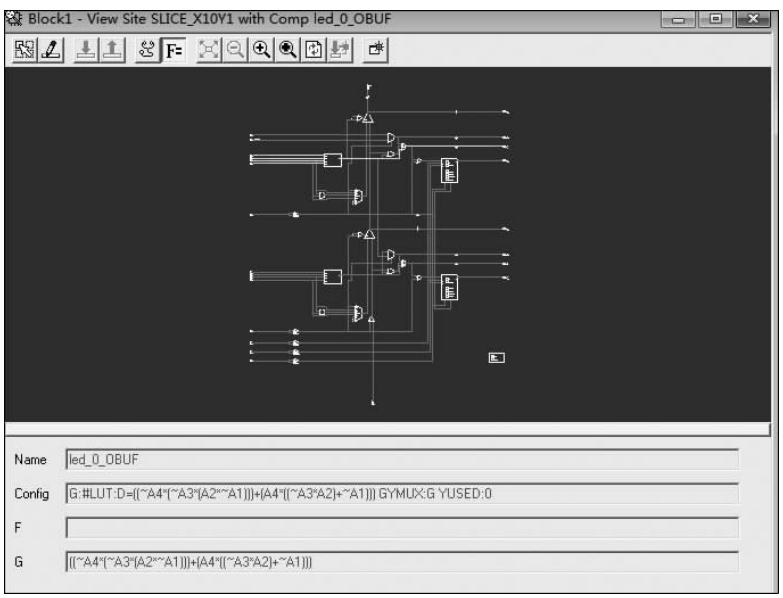


图 3-11　led[0]的实现方程

4. 拓展练习

在互联网上或通过其他途径查找 4 位数值比较器 74LS85 芯片的功能，然后使用 FPGA 予以实现该芯片，并在 Basys2 开发板上予以验证。