第3章 FPGA 开发综合实验

实验六：数码管显示

一、实验目的

（1）学习数码管的工作原理。

（2）实现对 4 位数码管的动态控制。

二、实验内容

动态控制 4 位数码管，使其能够正常工作。

三、实验要求

了解数码管动态显示数字的原理。

四、实验背景知识

1．数码管基础知识

数码管原理图如图 3.1 所示。

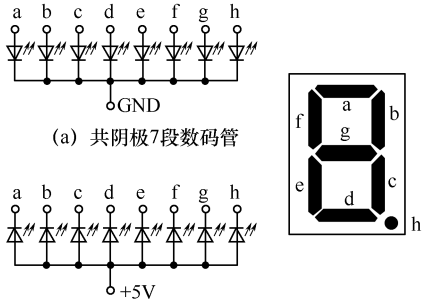


图 3.1　数码管原理图

数码管显示数字就是将相应的段位点亮进而组成要显示的数字，共阴极数码管对应的码值如表 3.1 所示。其中，“1”代表相应的引脚输出高电平，即点亮相应段位，“0”代表相应的引脚输出低电平，即不点亮相应段位。

表 3.1　共阴极数码管对应的码值

显示	a	b	c	d	e	f	g	dp(h)
0	1	1	1	1	1	1	0	0
1	0	1	1	0	0	0	0	0
2	1	1	0	1	1	0	1	0
3	1	1	1	1	0	0	1	0
4	0	1	1	0	0	1	1	0
5	1	0	1	1	0	1	1	0
6	1	0	1	1	1	1	1	0
7	1	1	1	0	0	0	0	0
8	1	1	1	1	1	1	1	0
9	1	1	1	1	0	1	1	0

2．数码管的工作原理

一般实验板上使用的是共阴极数码管，这种数码管有 4 个共阴极，分别用于选通对应的一位数码管，4 位数码管的 8 个段码引脚连接在一起，其硬件连接图如图 3.2 所示。

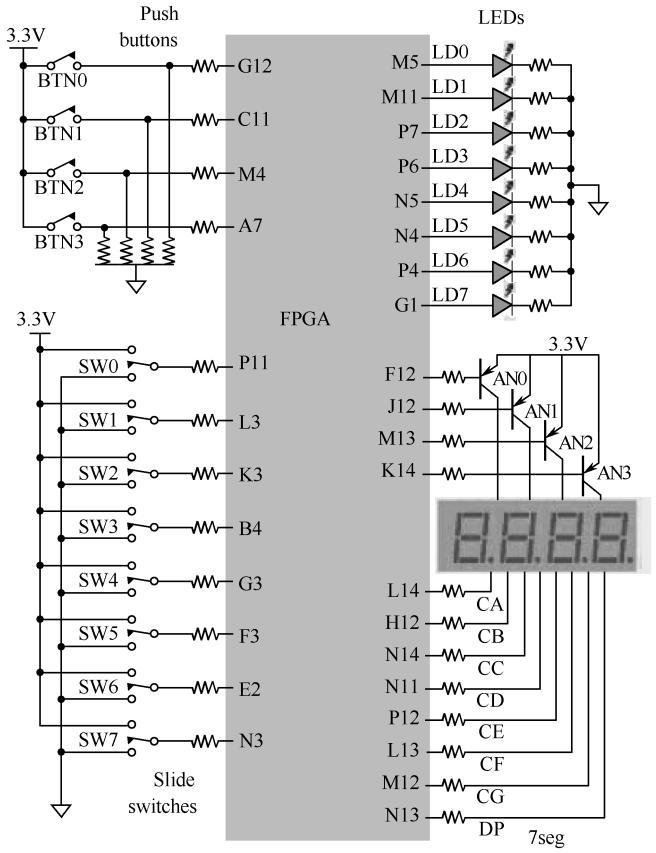


图 3.2　4 位数码管的硬件连接图

4 位数码管的工作原理是：每次选通其中一位数码管，输出这位数码管要显示的内容，然后一段时间后选通下一位数码管并输出对应数据，4 位数码管这样依次选通并输出相应的数据，结束后再重复进行。这样只要选通时间选取合适，并且由于人眼的视觉暂留现象，4 位数码管看起来是连续显示的。

五、实验步骤

（1）4 位数码管动态显示的设计共分为以下 3 个模块。

① 封装模块。

② 设计模块。

③ 顶层模块。

封装模块的程序如下。

```
//4 位数码管的 IP 核
module smg_ip_model(clk,data,sm_wei,sm_duan);
input clk;
input [15:0] data;
output [3:0] sm_wei;
output [7:0] sm_duan;
//-------------------------------------------------------------
//分频
integer clk_cnt;
reg clk_400Hz;
always @(posedge clk)
if(clk_cnt==32'd100000)
begin clk_cnt <= 1'b0; clk_400Hz <= ~clk_400Hz;end
else
clk_cnt <= clk_cnt + 1'b1;
//-------------------------------------------------------------
//位控制
reg [3:0]wei_ctrl=4'b1110;
always @(posedge clk_400Hz)
wei_ctrl <= {wei_ctrl[2:0],wei_ctrl[3]};
//段控制
reg [3:0]duan_ctrl;
always @(wei_ctrl)
case(wei_ctrl)
4'b1110:duan_ctrl=data[3:0];
4'b1101:duan_ctrl=data[7:4];
4'b1011:duan_ctrl=data[11:8];
4'b0111:duan_ctrl=data[15:12];
default:duan_ctrl=4'hf;
```

```
        endcase
        //-------------------------------------------------------------
        //解码模块
        reg [7:0]duan;
        always @(duan_ctrl)
        case(duan_ctrl)
        4'h0:duan=8'b0011_1111;//0
        4'h1:duan=8'b0000_0110;//1
        4'h2:duan=8'b0101_1011;//2
        4'h3:duan=8'b0100_1111;//3
        4'h4:duan=8'b0110_0110;//4
        4'h5:duan=8'b0110_1101;//5
        4'h6:duan=8'b0111_1101;//6
        4'h7:duan=8'b0000_0111;//7
        4'h8:duan=8'b0111_1111;//8
        4'h9:duan=8'b0110_1111;//9
        4'ha:duan=8'b0111_0111;//a
        4'hb:duan=8'b0111_1100;//b
        4'hc:duan=8'b0011_1001;//c
        4'hd:duan=8'b0101_1110;//d
        4'he:duan=8'b0111_1000;//e
        4'hf:duan=8'b0111_0001;//f
        default : duan = 8'b0011_1111;//0
        endcase
        //-------------------------------------------------------------

        assign sm_wei =~wei_ctrl;
        assign sm_duan = duan;
        endmodule
```

设计模块的程序如下。

```
        //测试 4 位数码管的 IP 核
        module test(clk,data);
        input clk;
        output [15:0]data;
        //-------------------------------------------------------------
        //分频 1Hz
        reg clk_1Hz;
        integer clk_1Hz_cnt;
```

```
always @(posedge clk)
if(clk_1Hz_cnt==32'd25000000-1)
begin clk_1Hz_cnt <= 1'b0; clk_1Hz <= ~clk_1Hz;end
else
clk_1Hz_cnt <= clk_1Hz_cnt + 1'b1;
//------------------------------------------------------------
//循环显示 0～9
reg [39:0]disp=40'h1234567890;
reg [15:0]data;
always @(posedge clk_1Hz)
begin
disp <= {disp[35:0],disp[39:36]};
data <= disp[39:24];
end
endmodule
```

顶层模块的程序如下。

```
//顶层模块
module smg_ip(clk,sm_wei,sm_duan);
input clk;
output [3:0]sm_wei;
output [7:0]sm_duan;
//------------------------------------------------------------
wire [15:0]data;
wire [3:0]sm_wei;
wire [7:0]sm_duan;
//------------------------------------------------------------
test U0 (.clk(clk),.data(data));
smg_ip_model U1 (.clk(clk),.data(data),.sm_wei(sm_wei),.sm_duan(sm_duan));
endmodule
```

（2）4 位数码管动态显示的电路图如图 3.3 所示。

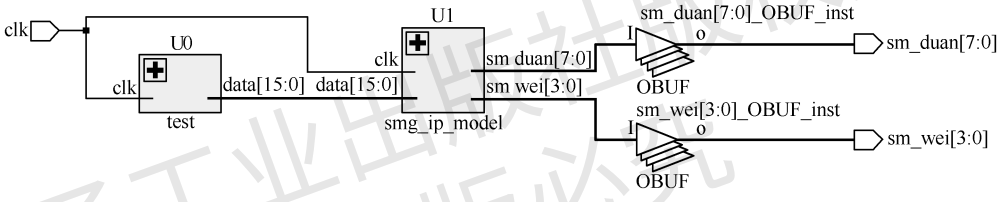


图 3.3　4 位数码管动态显示的电路图

六、实验结果

（1）编译已编写完毕的程序，程序编译通过后分配引脚，然后再次编译相关程序并生成下载文件。

（2）将程序下载到实验板上，通电后观察 4 位数码管的显示情况。若不能达到动态显示的效果，则需要检查源程序。待修改完毕后，再通电观察 4 位数码管的显示情况，直到正确显示为止。

实验七：交通灯的设计

一、实验目的

综合运用 Verilog HDL 进行交通灯的设计。

二、实验内容

（1）编写时间控制程序，并且利用交通灯实验板，实现东西向/南北向的交通灯计数并编写交通灯程序。

（2）利用交通灯实验板实现所有显示方面的功能，包括十进制数的倒计时和红、绿、黄三色灯的轮流点亮。

三、实验要求

（1）设置并修改两个方向上三色灯的点亮时间。

（2）交通灯控制器的状态转换表如表 3.2 所示。

表 3.2　交通灯控制器的状态转换表

东西向			南北向		
绿灯	黄灯	红灯	绿灯	黄灯	红灯
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

四、实验步骤

（1）为了在 8 位数码管上正确显示十进制数，需要设计一个函数，相关程序参考第 2 章实验五中的 3 个数码管模块的程序。数码管显示模块如图 3.4 所示。

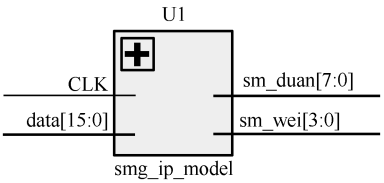


图 3.4　数码管显示模块

（2）主程序使用 case 语句，根据变量 Stage 的值确定当前状态，并加以执行。程序功能如下。

```
Stage=2'b00 表示南北方向绿灯点亮
Stage=2'b01 表示东西方向绿灯点亮
Stage=2'b10 表示南北方向绿灯转红灯期间的黄灯点亮
Stage=2'b11 表示东西方向绿灯转红灯期间的黄灯点亮
```

其中，绿灯点亮的时间为 20s；黄灯点亮的时间为 10s。

（3）主程序如下。

```
module test(clk,data,out_LED3_NS,out_LED3_WE);
input clk;
output [15:0]data;
output [2:0] out_LED3_NS,out_LED3_WE;
//分频 1Hz
reg clk_1Hz;
integer clk_1Hz_cnt;
always @(posedge clk)
if(clk_1Hz_cnt==32'd25000000-1)
begin clk_1Hz_cnt <= 1'b0; clk_1Hz <= ~clk_1Hz;end
else
clk_1Hz_cnt <= clk_1Hz_cnt + 1'b1;
//----------------------------------------------------
```

```
//循环显示 0～9
reg [15:0]data;
reg[2:0] out_LED3_NS,out_LED3_WE;
reg[3:0] Time_10=2'd2,Time_1=2'd0;
reg[1:0] Stage=2'b00;
always @(posedge clk_1Hz)
begin
case(Stage)
     2'b00:
     begin//NS pass
     if((Time_10==0) & (Time_1==0))
          begin
          Stage<=2'b11;
          Time_10<=4'd1;
          Time_1 <=4'd0;
          end
     else
          begin
          if(Time_1==0)
               begin
               Time_1<=4'd9;
               Time_10<=Time_10-1;
               end
          else
               begin
               Time_1<=Time_1-1;
               end
          end
     data[15:8]<={Time_10,Time_1};
     data[7:0]<={Time_10,Time_1};
     out_LED3_NS<=3'b001;
     out_LED3_WE<=3'b100;
     end
     2'b01:
     begin//WE pass
     if((Time_10==0) & (Time_1==0))
          begin
```

```
                Stage<=2'b10;
                Time_10<=4'd1;
                Time_1 <=4'd0;
                end
           else
                begin
                if(Time_1==0)
                     begin
                     Time_1<=4'd9;
                     Time_10<=Time_10-1;
                     end
                else
                     begin
                     Time_1<=Time_1-1;
                     end
                end
            data[15:8]<={Time_10,Time_1};
            data[7:0]<={Time_10,Time_1};
            out_LED3_WE<=3'b001;
            out_LED3_NS<=3'b100;
            end
            2'b10:
            begin//Yellow to NS pass
            if((Time_10==0) & (Time_1==0))
                 begin
                 Stage<=2'b00;
                 Time_10<=4'd2;
                 Time_1 <=4'd0;
                 end
            else
                 begin
                 if(Time_1==0)
                     begin
                     Time_1<=4'd9;
                     Time_10<=Time_10-1;
                     end
                 else
```

```
                begin
                Time_1<=Time_1-1;
                end
          end
     data[15:8]<={Time_10,Time_1};
     data[7:0]<={Time_10,Time_1};
     out_LED3_NS<=3'b010;
     out_LED3_WE<=3'b010;
     end
     2'b11:
     begin//Yellow to WE pass
     if((Time_10==0) & (Time_1==0))
          begin
          Stage<=2'b01;
          Time_10<=4'd2;
          Time_1 <=4'd0;
          end
     else
          begin
          if(Time_1==0)
              begin
              Time_1<=4'd9;
              Time_10<=Time_10-1;
              end
          else
              begin
              Time_1<=Time_1-1;
              end
          end
     data[15:8]<={Time_10,Time_1};
     data[7:0]<={Time_10,Time_1};
     out_LED3_NS<=3'b010;
     out_LED3_WE<=3'b010;
     end
     default:
          begin
          Stage<=2'b00;
```

```
                    Time_10<=4'd2;
                    Time_1<=4'd0;
                    end
        endcase
        end
        endmodule
```

（4）顶层文件的程序如下。

```
        module smg(clk,sm_wei,sm_duan,out_LED3_NS,out_LED3_WE);
        input clk;

        output [3:0]sm_wei;
        output [7:0]sm_duan;
        output [2:0] out_LED3_NS,out_LED3_WE;
        //------------------------------------------------------------
        wire [15:0]data;
        wire [3:0]sm_wei;
        wire [7:0]sm_duan;
        wire [2:0] out_LED3_NS,out_LED3_WE;
        //------------------------------------------------------------
        test U0 (.clk(clk),.data(data),.out_LED3_NS(out_LED3_NS),.out_LED3_WE(out_
LED3_WE));
        smg_ip_model U1 (.clk(clk),.data(data),.sm_wei(sm_wei),.sm_duan(sm_duan));
        endmodule
```

（5）总体交通灯电路图如图 3.5 所示。

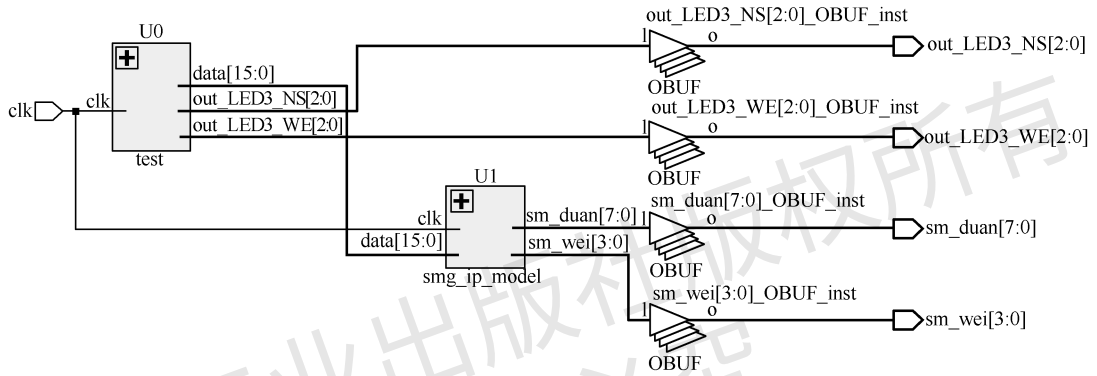


图 3.5　总体交通灯电路图

五、实验结果

（1）本实验是在交通灯实验板上实现的。
（2）将该实验板上电，将程序下载到 FPGA 芯片中。
（3）观察交通灯的显示结果。

实验八：秒表的设计

一、实验目的

（1）学习数码管的工作原理。
（2）实现对 4 位数码管的动态控制。
（3）熟悉模块化编程的设计过程。

二、实验内容

（1）动态控制 4 位数码管，使其能够正常工作。
（2）将 4 位数码管作为显示器件，并设计一个简单的秒表。

三、实验要求

（1）设计的秒表的最小计时单位为 0.1s。
（2）设计的秒表能够实现暂停和继续计时等功能。

四、实验步骤

（1）秒表的设计共分以下 3 个模块。
① 显示模块。
② 计时模块。
③ 顶层模块。

显示模块的程序参考本章实验六中的 4 位数码管模块的程序，在此不再赘述。计时模块的程序如下。

```
module time_counter(clk,key1,rst,data);
input clk;
input key1;
input rst;
output [15:0] data;
reg [15:0] data;
reg clk_1Hz;
integer clk_1Hz_cnt;
always @(posedge clk)
if(clk_1Hz_cnt==32'd25000000-1)
begin clk_1Hz_cnt <= 1'b0;
clk_1Hz <= ~clk_1Hz;
end
else
clk_1Hz_cnt <= clk_1Hz_cnt + 1'b1;
reg[3:0] time_1=0,time_10=0,time_100=0,time_1000=0;
always@(posedge clk_1Hz)
begin
     if(rst)
     begin
     time_1<=0;
     time_10<=0;
     time_100<=0;
     time_1000<=0;
     data[15:0] <={time_1000,time_100,time_10,time_1};
end
     else
          begin
               if(key1==0)
               begin
                    if(time_1 < 4'b1001)
                         begin
                         time_1 <= time_1+1'b1;
                         data[15:0] <={time_1000,time_100,time_10,time_1};
                         end
```

```
                    else if(time_10 < 4'b1001)
                         begin
                         time_10 <= time_10+1'b1;
                         time_1 <= 4'b0000;
                         data[15:0] <={time_1000,time_100,time_10,time_1};
                         end
                    else if(time_100 < 4'b1001)
                         begin
                         time_100 <= time_100+1'b1;
                         time_10 <= 4'b0000;
                         time_1 <= 4'b0000;
                         data[15:0] <={time_1000,time_100,time_10,time_1};
                         end
                    else if(time_1000 < 4'b1001)
                        begin
                        time_1000 <= time_1000+1'b1;
                        time_100    <=   4'b0000;
                        time_10 <=   4'b0000;
                        time_1 <= 4'b0000;
                        data[15:0] <={time_1000,time_100,time_10,time_1};
                        end
                    else
                        begin
                        time_1000 <= 4'b0000;
                        time_100    <=   4'b0000;
                        time_10 <=   4'b0000;
                        time_1 <= 4'b0000;
                        data[15:0] <={time_1000,time_100,time_10,time_1};
                        end
                    end
               else
                    begin
                    time_1000 <= time_1000;
                    time_100    <= time_100;
                    time_10     <= time_10;
                    time_1      <= time_1;
                    data[15:0] <={time_1000,time_100,time_10,time_1};
                    end
```

```
            end
        end
    endmodule
```

顶层模块的程序如下。

```
    module mb(clk,key1,rst,sm_wei,sm_duan);
    input clk;
    input key1;
    input rst;
    output [3:0]sm_wei;
    output [7:0]sm_duan;
    //-------------------------------------------------------------
    wire [15:0]data;
    wire [3:0]sm_wei;
    wire [7:0]sm_duan;
    //-------------------------------------------------------------
    time_counter U0 (.clk(clk),.rst(rst),.key1(key1),.data(data));
    smg_ip_model U1 (.clk(clk),.data(data),.sm_wei(sm_wei),.sm_duan(sm_duan));
     endmodule
```

（2）秒表总体设计电路图如图 3.6 所示。

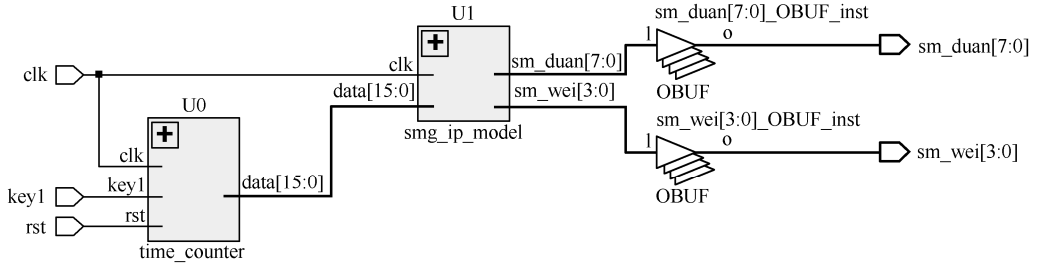


图 3.6　秒表总体设计电路图

五、实验结果

（1）编译编写完毕的程序，编译通过后，分配引脚，然后再次编译相关程序，生成下载文件。

（2）将程序下载到实验板上，通电后观察秒表的计数情况，按下开关 key1 后，

查看秒表是否停止计时，按下开关 key2 后，查看秒表是否继续计时。若不能达到实验效果，则需要继续检查源程序，修改好后再继续查看相关功能是否能实现，直到正确实现为止。

实验九：乐曲演奏实验

一、实验目的

（1）使用 FPGA 控制扬声器演奏《梁祝》中的一段。
（2）初步学会利用结构建模方法来设计程序。

二、实验内容

（1）利用时钟分频进行音调和音长的设定。
（2）利用整个程序演奏《梁祝》中的一段。
（3）令 LED 灯随音乐的节拍显示，分别显示高、中、低三种频率。

三、实验要求

（1）将时钟频率分成 5MHz 和 4Hz 两种。
（2）利用功能框图建立整个程序的设计流程，最终能够清晰地演奏乐曲。

四、实验背景知识

乐曲演奏的两个基本参数是每个音符的频率值（音调）及其持续的时间（音长），因此只要控制输出到扬声器的激励信号的频率和持续时间，就可以发出连续的声音。

1．音调控制

音符频率的高低决定音调的高低。简谱中，从低音 1 到高音 1 对应的频率如表 3.3 所示。

表 3.3　音名频率对照表

音名	频率/Hz	音名	频率/Hz	音名	频率/Hz
低音 1	261.6	中音 1	523.3	高音 1	1046.5
低音 2	293.7	中音 2	587.3	高音 2	1174.7
低音 3	329.6	中音 3	659.3	高音 3	1318.5
低音 4	349.2	中音 4	698.5	高音 4	1396.9
低音 5	392	中音 5	784	高音 5	1568
低音 6	440	中音 6	880	高音 6	1760
低音 7	493.9	中音 7	987.8	高音 7	1975.5

若基频过低，则由于分频比太小，会使四舍五入后的误差较大；若基频过高，则其误差会减小，但分频数会增大。综合以上两个因素考虑，选择 5MHz 作为基频。由于实验板上没有 5MHz 的时钟频率，因此必须先分频得到需要的频率。

《梁祝》的各音阶分频比与预置数如表 3.4 所示。

表 3.4 《梁祝》的各音阶分频比与预置数

音名	分频比	预置数	音名	分频比	预置数
低音 3	7585	8798	中音 2	4257	12126
低音 5	6378	10005	中音 3	3792	12591
低音 6	5682	10701	中音 5	3189	13194
低音 7	5062	11321	中音 6	2841	13542
低音 1	9557	6826	高音 1	2389	13994

为了减小输出的偶次谐波分量，输出到扬声器的波形应为对称方波，因此在扬声器前要加一个二分频。表 3.4 中给出了各音阶频率计数器不同的预置数。采用加载预置数实现分频的方法比采用反馈清零法节约资源，并且实现起来也相对容易。

对于乐曲中的休止符，将分频系数设为 0，即初始值为 $2\times10^{14}-1=16383$，此时扬声器不会发出声音。

2．音长控制

根据乐曲的速度及每个音符的节拍数来确定音符的持续时间。在《梁祝》中，最短的音符为四分音符，若将全音符的持续时间设为 1s，则需要再提供一个 4Hz 的时钟频率即可产生四分音符的时长。

乐曲演奏电路原理图如图 3.7 所示。

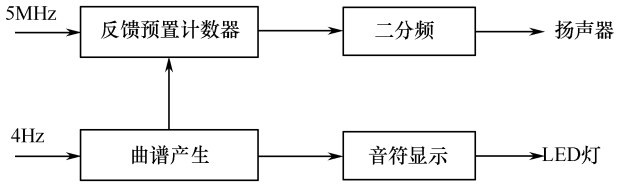


图 3.7　乐曲演奏电路原理图

3．乐曲演奏

乐曲演奏的主要源程序如下。

```
module song(clk,reset,speaker);
input clk;
input reset;
output speaker;
//output[3:0] high,med,low;
reg[3:0] high,med,low;
reg[13:0] divider,origin;
reg[7:0] counter;
reg[2:0] cnt;
reg[23:0] cnt1;
reg speaker;
wire carry;
assign carry=(divider==16383);
reg clk_5MHZ;
always@(posedge clk or posedge reset)
begin
if(reset)
      begin
      cnt<=0;
      clk_5MHZ<=0;
      end
else
      begin
      if(cnt==4)
           begin
           cnt<=0;
           clk_5MHZ<=~clk_5MHZ;
```

```
            end
        else
        cnt<=cnt+1;
        end
    end
    reg clk_4HZ;
    always@(posedge clk or posedge reset)
    begin
    if(reset)
        begin
        cnt1<=0;
        clk_4HZ<=0;
        end
    else
        begin
        if(cnt1==6249999)
            begin
            cnt1<=0;
            clk_4HZ<=~clk_4HZ;
            end
        else
            cnt1<=cnt1+1;
        end
    end
    always @(posedge clk_5MHZ)
    begin
    if(carry) divider=origin;
    else  divider=divider+1;
    end
    always @(posedge carry)
    begin
    speaker=~speaker;
    end
    always @(posedge clk_4HZ)
    begin
    case({high,med,low})
    'b000000000011: origin=8798;
    'b000000000101: origin=10005;
```

```
        'b000000000110: origin=10701;
        'b000000000111: origin=11321;
        'b000000010000: origin=6826;
        'b000000100000: origin=12126;
        'b000000110000: origin=12591;
        'b000001010000: origin=13194;
        'b000001100000: origin=13542;
        'b000100000000: origin=13994;
        'b000000000000: origin=16383;
        endcase
        end
        always @(posedge clk_4HZ)
        begin
        if(counter==63) counter=0;
        else counter=counter+1;
        case(counter)
        0:{high,med,low}='b000000000011;
        1:{high,med,low}='b000000000011;
        2:{high,med,low}='b000000000011;
        3:{high,med,low}='b000000000011;
        4:{high,med,low}='b000000000101;
        5:{high,med,low}='b000000000101;
        6:{high,med,low}='b000000000101;
        7:{high,med,low}='b000000000110;
        8:{high,med,low}='b000000010000;
        9:{high,med,low}='b000000010000;
        10:{high,med,low}='b000000010000;
        11:{high,med,low}='b000000100000;
        12:{high,med,low}='b000000000110;
        13:{high,med,low}='b000000010000;
        14:{high,med,low}='b000000000101;
        15:{high,med,low}='b000000000101;
        16:{high,med,low}='b000001010000;
        17:{high,med,low}='b000001010000;
        18:{high,med,low}='b000001010000;
        19:{high,med,low}='b000100000000;
        20:{high,med,low}='b000001100000;
        21:{high,med,low}='b000001010000;
```

```
        22:{high,med,low}='b000000110000;
        23:{high,med,low}='b000001010000;
        24:{high,med,low}='b000000100000;
        25:{high,med,low}='b000000100000;
        26:{high,med,low}='b000000100000;
        27:{high,med,low}='b000000100000;
        28:{high,med,low}='b000000100000;
        29:{high,med,low}='b000000100000;
        30:{high,med,low}='b000000100000;
        31:{high,med,low}='b000000100000;
        32:{high,med,low}='b000000100000;
        33:{high,med,low}='b000000100000;
        34:{high,med,low}='b000000100000;
        35:{high,med,low}='b000000110000;
        36:{high,med,low}='b000000000111;
        37:{high,med,low}='b000000000111;
        38:{high,med,low}='b000000000110;
        39:{high,med,low}='b000000000110;
        40:{high,med,low}='b000000000101;
        41:{high,med,low}='b000000000101;
        42:{high,med,low}='b000000000101;
        43:{high,med,low}='b000000000110;
        44:{high,med,low}='b000000010000;
        45:{high,med,low}='b000000010000;
        46:{high,med,low}='b000000100000;
        47:{high,med,low}='b000000100000;
        48:{high,med,low}='b000000000011;
        49:{high,med,low}='b000000000011;
        50:{high,med,low}='b000000010000;
        51:{high,med,low}='b000000010000;
        52:{high,med,low}='b000000000110;
        53:{high,med,low}='b000000000101;
        54:{high,med,low}='b000000000110;
        55:{high,med,low}='b000000010000;
        56:{high,med,low}='b000000000101;
        57:{high,med,low}='b000000000101;
        58:{high,med,low}='b000000000101;
        59:{high,med,low}='b000000000101;
```

```
        60:{high,med,low}='b000000000101;
        61:{high,med,low}='b000000000101;
        62:{high,med,low}='b000000000101;
        63:{high,med,low}='b000000000101;
        endcase

        end endmodule
```

五、实验结果

（1）编译编写完毕的程序，编译通过后，分配引脚，然后再次编译相关程序并生成下载文件。

（2）将程序下载到实验板上，通电后检查扬声器发出的声音是否正确。

实验十：VGA

一、实验目的

（1）了解 VGA 显示的基本原理和实现方法。

（2）学会使用 Verilog HDL 语言编写 VGA 显示的接口程序。

二、实验内容

（1）掌握 VGA 通信的基本原理和实现方法。

（2）理解 VGA 接口程序的实现方法和使用方法。

三、实验要求

掌握已经编写好的 VGA 程序，根据实验提供的思路自主编写 VGA 显示的例程。

四、实验背景知识

1．VGA 的概念

VGA（Video Graphic Array，视频图形阵列）支持在 640×480 的较高分辨率下同时显示 16 种色彩或 256 种灰度，并且在 320×240 分辨率下可以同时显示 256 种颜色。人的肉眼对颜色的敏感度远高于分辨率，所以即使分辨率较低的图像在人们看来依然清晰、鲜明。由于 VGA 具有良好的性能，因此其使用范围很广泛，各大厂商纷纷在 VGA 基础上加以扩充，如将显存提高至 1MB 并使其支持更高的分辨率（如 800×600 或 1024×768），这些扩充的模式称为 VESA（Video Electronics Standards Association，视频电子标准协会）的 Super VGA 模式，简称 SVGA。现在的显卡和显示器都支持 SVGA 模式。无论是 VGA 还是 SVGA，使用的接口都是 15 针的梯形插头，用于传输模拟信号。

2．VGA 的接口信号

目前，大多数计算机与外部显示设备之间都是通过模拟 VGA 接口连接的，计算机内部以数字方式生成的显示图像信息，被显卡中的数字/模拟（D/A）转换器转换为 R、G、B 三原色信号和行/场同步信号，这些信号通过电缆传输到显示设备中。本实验中，VGA 接口是标准的 15 针接口，具有 5 个接口信号，VGA 接口信号如表 3.5 所示。

表 3.5　VGA 接口信号

信号	信号含义
HS	行同步信号
VS	场同步信号
R	红色信号
G	绿色信号
B	蓝色信号

3．行同步和场同步

为了实现发送端与接收端图像各点正确对应，发送端与接收端的扫描必须同步。同步脉冲是周期稳定、边沿陡峭的脉冲。根据我国数字电视标准，行同步脉冲的频率为 15.625kHz，行周期为 64μs。常以 64μs 作为时间单位，并用 H 表示，即 1H=64μs。场同步脉冲频率为 50Hz，场周期为 20ms，即 312.5H。行同步脉冲宽度为

4.7μs 左右，场同步脉冲宽度为 160～192μs，即 2.5H～3H。

五、实验步骤

主程序中有 4 个输入信号和 5 个输出信号，其中两个输入信号分别为 clock 和 switch。clock 是时钟信号，switch 是选择模式信号，分别对应横彩条、竖彩条及两种棋盘格。输出信号在前面已经介绍过了，这里不再赘述。

VGA 源程序如下。

```
module vga( clock，switch，disp_RGB，hsync，vsync );
input clock; //系统输入时钟为 100MHz
input [1:0]switch;
output [2:0]disp_RGB; //VGA 数据输出
output hsync; //VGA 行同步信号 output
vsync; //VGA 场同步信号
reg [9:0] hcount; //VGA 行扫描计数器
reg [9:0] vcount; //VGA 场扫描计数器
reg [2:0] data;
reg [2:0] h_dat;
reg [2:0] v_dat;
reg flag;
reg [1:0]cnt;
wire hcount_ov;
wire vcount_ov;
wire dat_act;
wire hsync;
wire vsync;
reg vga_clk;
//VGA 行/场扫描时序参数表
parameter hsync_end = 10'd95,
hdat_begin = 10'd143,
hdat_end = 10'd783,
hpixel_end = 10'd799,
vsync_end = 10'd1,
vdat_begin = 10'd34,
vdat_end = 10'd514,
vline_end = 10'd524;
```

```
always @(posedge clock)
begin
if(cnt==3)
     cnt <= 0;
   else
     cnt <= cnt + 1;
end
always @(posedge clock)
begin
  if(cnt < 2)
     vga_clk <= 1;
   else
     vga_clk <= 0;
end
//************************VGA 驱动部分*******************************//
//行扫描
always @(posedge vga_clk)
begin
if (hcount_ov)
hcount <= 10'd0;
else
hcount <= hcount + 10'd1;
end
assign hcount_ov = (hcount == hpixel_end);
//场扫描
always @(posedge vga_clk)
begin
if (hcount_ov)
begin
if (vcount_ov)
vcount <= 10'd0;
else
vcount <= vcount + 10'd1;
end
end
assign vcount_ov = (vcount == vline_end);
//数据信号与同步信号输入
assign dat_act = ((hcount >= hdat_begin) && (hcount < hdat_end))
```

```
                && ((vcount >= vdat_begin) && (vcount < vdat_end));
    assign hsync = (hcount > hsync_end);
    assign vsync = (vcount > vsync_end);
    assign disp_RGB = (dat_act) ? data : 3'h00;
    //************************显示数据处理部分*******************************//
    always @(posedge vga_clk)
      begin
        case(switch[1:0])
          2'd0: data <= h_dat; //选择横彩条
          2'd1: data <= v_dat; //选择竖彩条
          2'd2: data <= (v_dat ^ h_dat); //产生棋盘格
          2'd3: data <= (v_dat ~^ h_dat); //产生棋盘格
        endcase
      end
    always @(posedge vga_clk) //产生竖彩条
      begin
        if(hcount < 223)
          v_dat <= 3'h7; //白
        else if(hcount < 303)
          v_dat <= 3'h6; //黄
        else if(hcount < 383)
          v_dat <= 3'h5; //青
        else if(hcount < 463)
          v_dat <= 3'h4; //绿
        else if(hcount < 543)
          v_dat <= 3'h3; //紫
        else if(hcount < 623)
          v_dat <= 3'h2; //红
        else if(hcount < 703)
          v_dat <= 3'h1; //蓝
        else
          v_dat <= 3'h0; //黑
      end
    always @(posedge vga_clk) //产生横彩条
      begin
        if(vcount < 94)
          h_dat <= 3'h7; //白
        else if(vcount < 154)
```

```
            h_dat <= 3'h6; //黄
          else if(vcount < 214)
            h_dat <= 3'h5; //青
          else if(vcount < 274)
            h_dat <= 3'h4; //绿
          else if(vcount < 334)
            h_dat <= 3'h3; //紫
          else if(vcount < 394)
            h_dat <= 3'h2; //红
          else if(vcount < 454)
            h_dat <= 3'h1; //蓝
          else
            h_dat <= 3'h0; //黑
        end
      endmodu
```

六、实验结果

（1）编译已经编写完毕的程序，编译通过后，分配引脚，然后再次编译相关程序，并生成下载文件。

（2）将程序下载到实验板上，接好 VGA 线，观察显示内容。

实验十一：PS/2 接口控制

一、实验目的

（1）了解 PS/2 键盘的工作原理。

（2）学会使用 Verilog HDL 语言编写 PS/2 接口的控制程序。

二、实验内容

（1）通过 Verilog HDL 编程实现在实验板上对 PS/2 键盘接口的控制，通过键盘输入实现对 LED 灯的控制（按下 W 键时，LED 灯向右依次点亮，按下 X 键时，LED 灯向左依次点亮，按下 Ctrl 键时左右方向交换点亮）。

（2）理解 PS/2 接口的控制程序的实现方法和使用方法。

三、实验要求

理解已经编写好的 PS/2 程序，根据实验思路自主编写 PS/2 在 LCD 或者 VAG 上的显示例程。

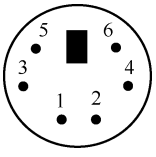
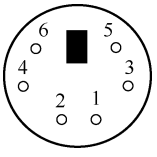
四、实验背景知识

1．PS/2 键盘协议和扫描码

（1）PS/2 键盘遵守双向同步串行协议。即数据线上每发送一位数据，并且在时钟线上每发送一个脉冲就被读入。键盘可以发送数据到主机，而主机也可以发送数据到设备。但主机总是在总线上有优先权，它可以在任意时刻抑制来自键盘的通信。

本实例要编写一个能实现 PS/2 接口功能的 Verilog HDL 程序。首先需要了解 PS/2 接口的结构与引脚功能定义，如表 3.6 所示。

表 3.6　PS/2 接口结构及引脚功能定义

公插	母插	6-pin Mini-DIN (PS/2)	6 脚 Mini-DIN（PS/2）
		1-Data	1—数据
		2-Not Implemented	2—未实现，保留
		3-Ground	3—电源地
		4-+5V	4—电源+5V
（Plug）插头	（Socket）插座	5-Clock	5—时钟
		6-Not Implemented	6—未实现，保留

由表 3.6 可以看出，PS/2 只有一个数据口，若要利用这一个数据口来分辨多个按键就需要一种高效率的分辨方法。键盘处理器花费很长时间来扫描或监视按键矩阵，若发现有按键被按下或被释放，则会扫描码的信息包发送到计算机上。PS/2 时序图如图 3.8 所示。

当 PS/2 设备需要传输数据时，会在 PS/2 时钟（CLOCK）的第一个下降沿拉低 PS/2 数据线（DATA），表示一帧 PS/2 数据传输的开始。接着传输 8 个数据位（都是在时钟的下降沿有效），然后传输奇偶校验位，最后传输停止位（高电平）。主机在接收完一帧数据后，需要将接收到的串行数据转换成并行数据，这就是 PS/2 键盘扫描码。

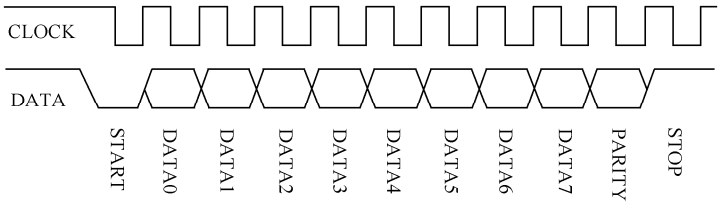


图 3.8　PS/2 时序图

（2）PS/2 键盘扫描码分为两种不同的类型：通码（Make Code）和断码（Break Code）。当按下按键或持续按下按键时，键盘会发送该按键的通码；而当一个键被释放时，键盘会发送该按键的断码。根据键盘按键扫描码的不同，可将按键分为以下 3 类。

① 第一类按键：通码为 1 字节，断码为 0xF0+通码的形式。如 A 键，其通码为 0x1C，断码为 0xF0 0x1C。

② 第二类按键：通码为 2 字节，其形式为 0xE0+0xXX，断码为 0xE0+0xF0+0xXX 的形式。如 Ctrl 键，其通码为 0xE0 0x14，断码为 0xE0 0xF0 0x14。

③ 第三类特殊按键：该类按键有两个，即 Print Screen 键，其通码为 0xE0 0x12 0xE0 0x7C，断码为 0xE0 0xF0 0x7C 0xE0 0xF0 0x12；Pause 键，其通码为 0xE1 0x14 0x77 0xE1 0xF0 0xl4 0xF0 0x77，断码为空。

每个按键对应的通码/断码组成了扫描码集，图 3.9 中包含了键盘上大部分按键的扫描码。

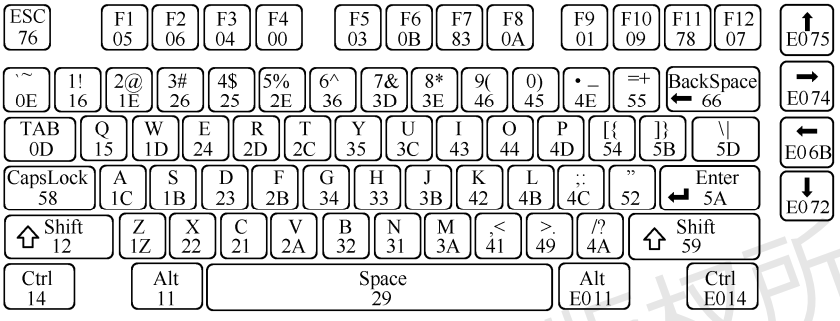


图 3.9　键盘上大部分按键的扫描码

（3）信号通过键盘及 PS/2 接口数据线的输入过程。首先，由键盘检测数据线和时钟线是否均为高电平，若两者都处在高电平，则可以写数据。从键盘发送到主机的数据在时钟信号的下降沿（当时钟从高电平变为低电平）被读取。键盘主要遵循一种

每帧包含 11 位的串行协议，即第一位是起始位，始终为“0”；接下来是 8 位数据位，排列顺序由低到高，再后面是奇偶校验位，最后是结束位，该位始终为“1”。

2．PS/2 键盘解码模块

PS/2 键盘事件响应流程如图 3.10 所示。

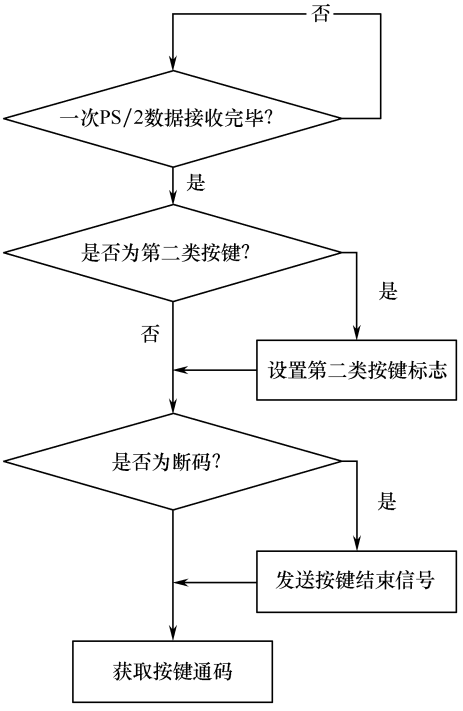


图 3.10　PS/2 键盘事件响应流程

当有按键按下时，PS/2 解码模块先将按键通码保存，然后一直等待按键松开，直到接收按键断码，此时一次按键事件结束。接着，按照第二套扫描码集的规定，将按键通码转换为 ASCII 码，并产生一个按键事件信号，等待其他模块处理。

五、实验步骤

程序设计共分为以下 4 个模块。

（1）PS/2 接口模块的程序如下。

```
module keyboard(
```

```
        input wire clk_25MHz,
        input wire clr,
        input wire PS2C,
        input wire PS2D,
        output wire[15:0]xkey
        );
        reg PS2Cf, PS2Df;
        reg [7: 0]ps2c_filter, ps2d_filter;
        reg[10:0]shift1,shift2;
        assign xkey = {shift2[8:1],shift1[8:1]};
        //filter for PS2 clock and data
        always @(posedge clk_25MHz or posedge clr)
          begin
          if(clr== 0)
             begin
             ps2c_filter<=0;
             ps2d_filter<=0;
             PS2Cf<=1;
             PS2Df<= 1;
             end
          else
             begin
             ps2c_filter[7]<=PS2C;
             ps2c_filter[6: 0]<=ps2c_filter[7: 1];
             ps2d_filter[7]<=PS2D;
             ps2d_filter[6: 0] <=ps2d_filter[ 7: 1];
             if(ps2c_filter ==8'b11111111)
                PS2Cf<=1;
             else if(ps2c_filter ==8'b00000000)
                PS2Cf<=0;
             if(ps2d_filter ==8'b11111111)
                PS2Df<=1;
             else if(ps2d_filter ==8'b00000000)
                PS2Df<=0;
             end
          end
        //Shift register used to clock in scan codes from PS2
        always @(negedge PS2Cf or posedge clr)
```

```
    begin
    if(clr==0)
       begin
       shift1 <=0;
       shift2 <=1;
       end
    else
       begin
       shift1 <= {PS2Df, shift1[10:1]};
       shift2 <= {shift1[0],shift2[10:1]};
       end
    end
endmodule
```

（2）键盘接口顶层模块的程序如下。

```
module keyboard_top(
input wire clk_100MHz,
input wire PS2C,
input wire PS2D,
input wire clr,
output wire [6: 0]a_to_g,
output wire[3:0]an
);
wire pclk, clk_25MHz;
wire [15:0] xkey;
clkdiv U1(
  .clk_100MHz(clk_100MHz),
  .clr(clr),
  .clk_25MHz(clk_25MHz)
  );
keyboard U2(
  .clk_25MHz(clk_25MHz),
  .clr(clr),
  .PS2C(PS2C),
  .PS2D(PS2D),
  .xkey(xkey)
  );
x7seg U3(
```

```
    .x(xkey),
    .clk(clk_100MHz),
    .clr(clr),
    .a_to_g(a_to_g),
    .an(an)
    );
endmoduleendmodule
```

（3）7 段数码管显示模块的程序如下。

```
module x7seg(
  input wire [15:0] x,
  input wire clk,
  input wire clr,
  output reg [6:0]a_to_g,
  output reg [3:0]an
  );
  wire[1:0]clk_190Hz;
  reg[3:0]digit;
  reg[19:0]clkdiv;
  assign clk_190Hz=clkdiv[19:18];
  always@(*)
   case(clk_190Hz)
    0:digit=x[3:0];
    1:digit=x[7:4];
    2:digit=x[11:8];
    3:digit=x[15:12];
    default:digit=x[3:0];
   endcase
  always@(*)
   case(digit)
    0:a_to_g=7'b1111110;
    1:a_to_g=7'b0110000;
    2:a_to_g=7'b1101101;
    3:a_to_g=7'b1111001;
    4:a_to_g=7'b0110011;
    5:a_to_g=7'b1011011;
    6:a_to_g=7'b1011111;
    7:a_to_g=7'b1110000;
```

```
            8:a_to_g=7'b1111111;
            9:a_to_g=7'b1111011;
            'hA:a_to_g=7'b1110111;
            'hB:a_to_g=7'b0011111;
            'hC:a_to_g=7'b1001110;
            'hD:a_to_g=7'b0111101;
            'hE:a_to_g=7'b1001111;
            'hF:a_to_g=7'b1000111;
             default:a_to_g=7'b1111110;
          endcase
        always@(*)
          begin
           an=4'b0000;
            an[clk_190Hz]=1;
          end
        always@(posedge clk or posedge clr)
          begin
           if(clr==0)
            clkdiv<=0;
           else
            clkdiv<=clkdiv+1;
          end
      endmodule
```

（4）时钟分频模块的程序如下。

```
      module clkdiv(
         input wire clk_100MHz,
         input wire clr,
         output wire clk_25MHz
        );

      reg [21:0]q;
      //25 位计数器
      always@(posedge clk_100MHz or posedge clr)
      begin
        if(clr==0)
          q<=0;
        else
```

```
        q<=q+1;
    end
    assign clk_25MHz=q[1];
    endmodule
```

六、实验结果

（1）编译编写好的程序，编译通过后，分配引脚，然后再次编译相关程序，生成下载文件。

（2）将程序下载到实验板上，通过扩展接口连接 PS/2 接口线，按下 W 键、X 键或 Ctrl 键查看 LED 灯点亮或熄灭的情况（由于按键存在抖动，因此会存在按一下按键，LED 灯变化两次的情况）。

实验十二：系统 IP 核的调用

一、实验目的

Vivado 中有很多 IP 核可以直接使用，如数学运算（DSP、乘法器、除法器、浮点运算器等）、信号处理（FFT、DFT、DDS 等）。IP 核类似程序中的函数库（如 C 语言中的 printf()函数），可以直接调用，大大加快了开发速度，本实验目的是能够熟练掌握系统 IP 核的调用。

二、实验内容

实现系统 IP 核的调用。

三、实验要求

以乘法器的 IP 核使用为例，熟练掌握系统 IP 核的调用。

四、实验步骤

1．创建 IP 核

以一个简单的乘法器的 IP 核使用为例，利用 Verilog HDL 调用 IP 核。首先，新建一个工程，然后新建 demo.v 顶层模块。

（1）单击“Flow Navigator”列表框中的“IP Catalog”选项。依次选择“Math Functions”→“Multipliers”→“Multiplier”选项，即乘法器，并双击鼠标右键。如图 3.11 所示。

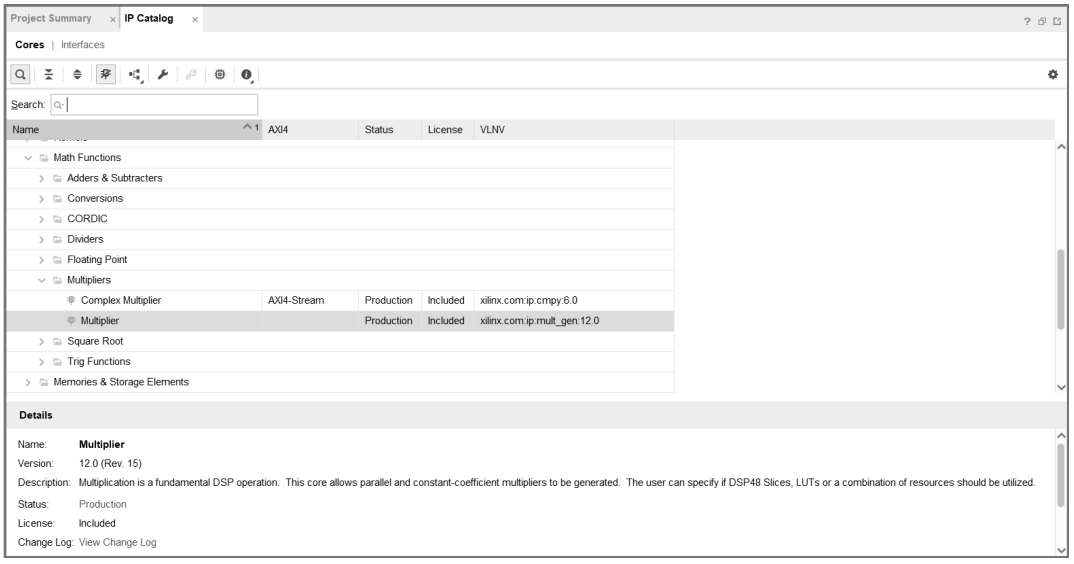


图 3.11　“选择乘法器”界面

（2）弹出“Customize IP”窗口。单击左上角的“Documentation”按钮，查阅该 IP 核的使用手册。在“Input Options”工作区中直接将输入信号 A 和输入信号 B 均设置为 4 位无符号型数据，其他选项均为默认设置，然后单击“OK”按钮，如图 3.12 所示。

（3）然后弹出“Generate Output Products”窗口，单击“Out of context per IP”单选按钮，然后单击“Generate”按钮，如图 3.13 所示。

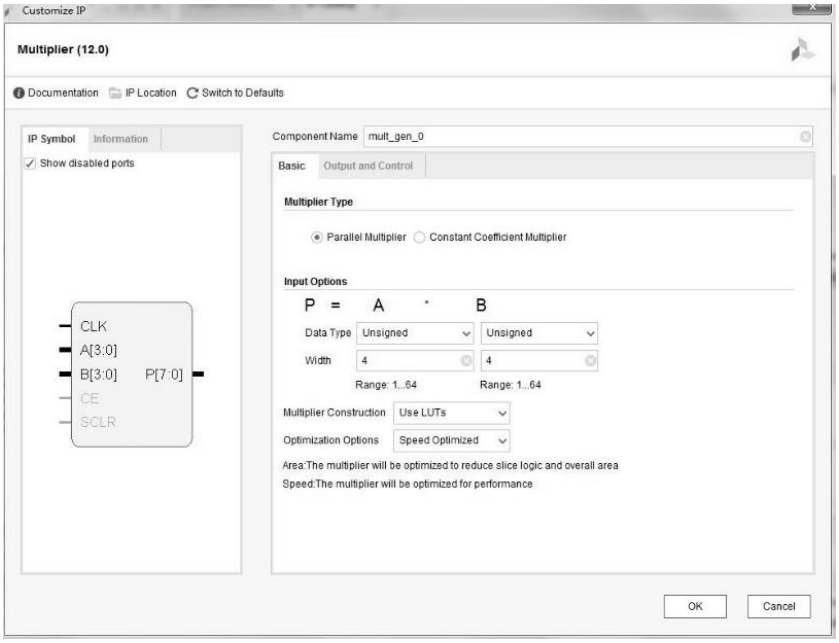


图 3.12 “Customize IP”窗口

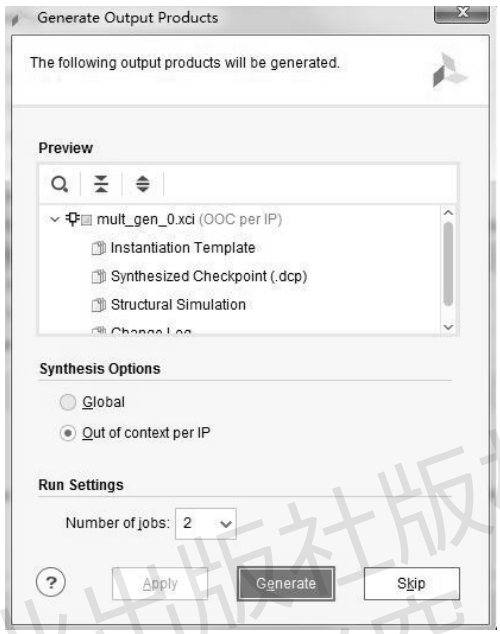


图 3.13 “Generate Output Products”窗口

2．调用 IP 核

在“Sources”工作区中，展开“mult_gen_0”选项，打开“mult_gen_0.veo”文件，打开实例化模板文件。如图 3.14 所示，第 56 行代码就是使用 Verilog HDL 语言调用该 IP 核的示例代码。

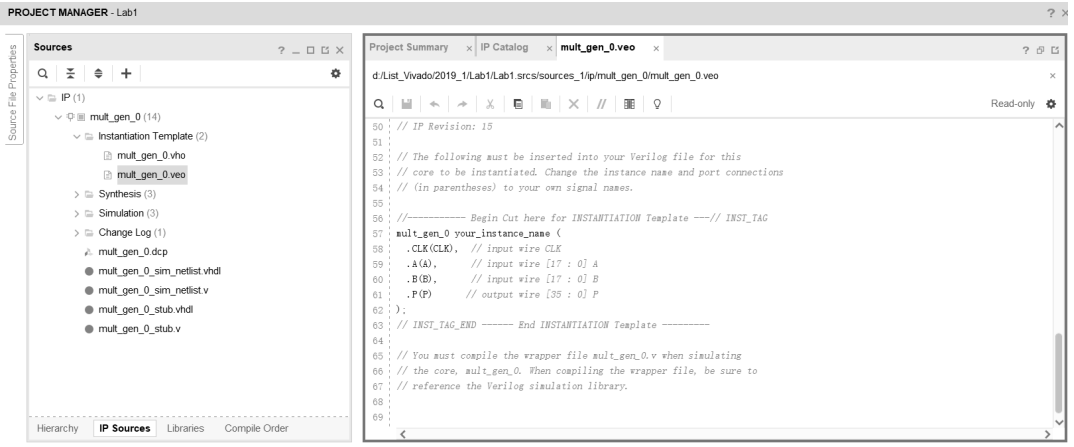


图 3.14　IP 核实例化模板文件

3．仿真验证

将示例代码复制到 demo.v 文件中，并对其进行如下修改。

```
module demo(
);
reg clk = 0;
always #10 clk = ~clk;
wire [3:0] a = 7;
wire [3:0] b = 8;
wire [7:0] p;
mult_gen_0 mul (
.CLK(clk), //input wire CLK
.A(a),     //input wire [3 : 0]A
.B(b),     //input wire [3 : 0]B
.P(p)      //output wire [7 : 0] P
);
endmodule
```

其中，声明了无符号型的 4 位变量 a 和 b，分别赋初值 7 和 8，并作为乘数使用；无符号型的 8 位变量 p，用于保存计算结果。clk 是周期为 20ns 的时钟信号；mult_gen_0 mul(...)语句实例化了 mult_gen_0 类型的模块对象 mul，并将 clk、a、b、p 作为参数传入。

五、实验结果

以 demo 为顶层模块，启动行为仿真，即可输出波形。设置 a、b、p 显示为无符号十进制数（单击鼠标右键选择“Radix-Unsigned Decimal”选项）。如图 3.15 所示，可以看到 a=7,b=8，第一个时钟上升沿后 p = a × b = 56。

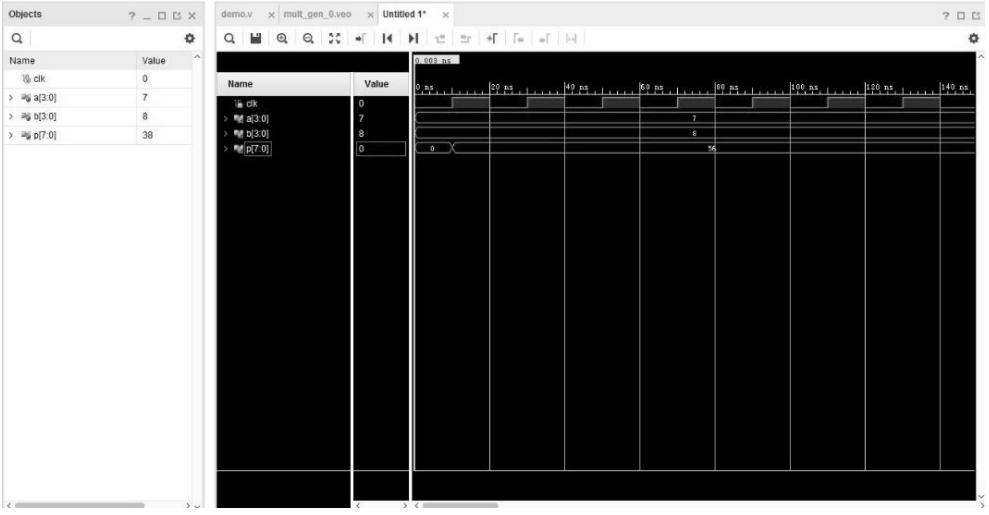


图 3.15　乘法器 IP 核仿真波形

实验十三：手机蓝牙通信实验

一、实验目的

（1）熟悉 UART 串口的使用方法。

（2）了解运用 Verilog HDL 语言开发蓝牙串口的方式。

二、实验内容

（1）通过 Verilog HDL 编程在实验板上设计蓝牙串口，令支持蓝牙 4.0 的手机与实验板上的蓝牙模块建立连接，并且通过手机 App 发送命令，进而控制实验板上的硬件外设。

（2）理解蓝牙串口控制程序的实现方法和使用方法。

三、实验要求

本实验中，利用实验板上的蓝牙模块与外界支持蓝牙 4.0 标准的设备（如手机）进行交互。该蓝牙模块默认配置为通过串口协议与 FPGA 进行通信，用户无须了解蓝牙的相关协议与标准，只需按照 UART 串口通信协议来处理发送与接收数据即可。

四、实验步骤

蓝牙无线技术是使用范围最广的全球短距离无线标准之一。本实验以 TI 公司 BLE-CC41-A 蓝牙 4.0 模块为例，该模块具有 256KB 配置空间，遵循蓝牙 BLE 4.0 标准。BLE-CC41-A 蓝牙 4.0 模块电路图如图 3.16 所示。

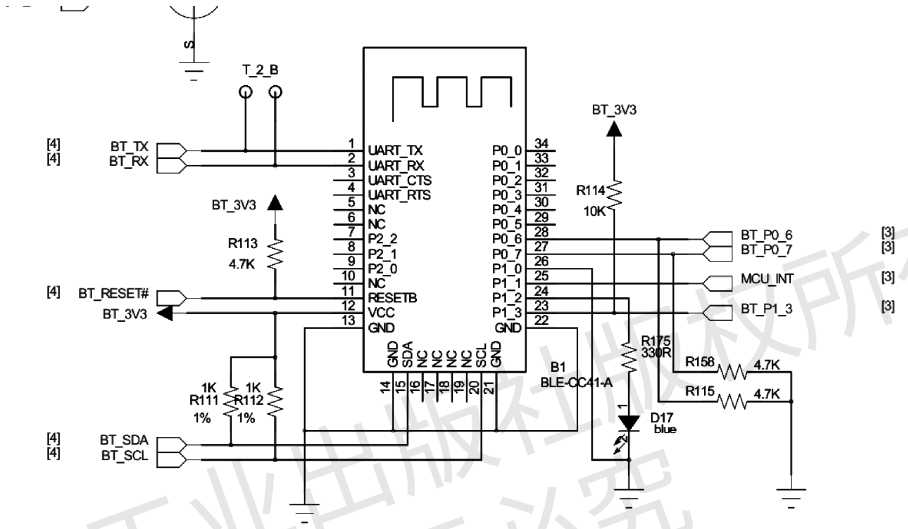


图 3.16　BLE-CC41-A 蓝牙 4.0 模块电路图

编写 uart_top 模块的程序，该模块提供了顶层接口，包括对外数据通路 rx 和 tx、蓝牙配置部分及外部时钟输入。

（1）顶层文件的程序如下。

```
module uart_top(
  output txd,
  input rxd,
  input clk,
  output bt_pw_on,
  output bt_master_slave,
  output bt_sw_hw,
  output bt_rst_n,
  output bt_sw,
  input [5:0] sw_pin
  );
  wire clk_9600;
  wire receive_ack;
  wire [7:0] data;
  uart_tx uart_tx(
  .clk(clk_9600),
  .txd(txd),
  //.rst(1),
  .data_o(data),
  .receive_ack(receive_ack)
  );
  uart_rx uart_rx(
  .clk(clk_9600),
  .rxd(rxd),
  .data_i(data),
  .receive_ack(receive_ack)
  );
  clk_div clk_div(
  .clk(clk),
  .clk_out(clk_9600)
  );
  assign bt_master_slave = sw_pin[0];
  assign bt_sw_hw        = sw_pin[1];
  assign bt_rst_n        = sw_pin[2];
```

```
  assign bt_sw           = sw_pin[3];
  assign bt_pw_on        = sw_pin[4];
endmodule
```

（2）串口发送模块的程序如下。

```
module uart_tx(
input [7:0]data_o,
output reg txd,
input clk,
//input rst,
input receive_ack
);
//串口发送状态机分为 4 个状态：等待、发送起始位、发送数据、发送完成
localparam IDLE = 0,
       SEND_START = 1,
       SEND_DATA = 2,
       SEND_END = 3;
reg [3:0] cur_st,nxt_st;
reg [4:0] count;
reg [7:0] data_o_tmp;

always @(posedge clk)
  begin
   cur_st <=  nxt_st;
  end
always @(*)
  begin
    nxt_st =  cur_st;
    case(cur_st)
      IDLE: if(receive_ack) nxt_st = SEND_START;
      SEND_START: nxt_st = SEND_DATA;
      SEND_DATA: if(count == 7) nxt_st = SEND_END;
      SEND_END: if(receive_ack) nxt_st = SEND_START;
      default: nxt_st = IDLE;
    endcase
  end

  always @(posedge clk)
```

```
  begin
    if(cur_st == SEND_DATA)
       count <= count + 1;
    else if(cur_st == IDLE|cur_st== SEND_END)
       count <= 0;
  end

 always @(posedge clk)
  begin
    if(cur_st == SEND_START)
       data_o_tmp <= data_o;
    else if(cur_st == SEND_DATA)
       data_o_tmp[6:0] <= data_o_tmp[7:1];
  end

always @(posedge clk)
   begin
     if(cur_st == SEND_START)
        txd <= 0;
      else if(cur_st == SEND_DATA)
        txd <= data_o_tmp[0];
      else if(cur_st == SEND_END)
        txd <= 1;
   end

endmodule
```

（3）串口接收模块的程序如下。

```
module uart_rx(
  input rxd,
  input clk,
  output receive_ack,
  output reg[7:0]data_i
  );
  //串口接收状态机分为 3 个状态：等待、接收、接收完成
  localparam IDLE = 0,
       RECEIVE = 1,
       RECEIVE_END = 2;
```

```
    reg [3:0] cur_st,nxt_st;
    reg [4:0] count;
    reg [7:0] data_o_tmp;

    always @(posedge clk)
      begin
        cur_st <=  nxt_st;
      end
    always @(*)
      begin
        nxt_st =  cur_st;
        case(cur_st)
          IDLE: if(!rxd) nxt_st = RECEIVE;
          RECEIVE: if(count == 7) nxt_st = RECEIVE_END;
          RECEIVE_END: nxt_st = IDLE;
          default: nxt_st = IDLE;
        endcase
      end

    always @(posedge clk)
      begin
        if(cur_st == RECEIVE)
           count <= count + 1;
        else if(cur_st == IDLE|cur_st== RECEIVE_END)
           count <= 0;
      end

    always @(posedge clk)
      begin
        if(cur_st == RECEIVE)
           begin
             data_i[6:0] <= data_i[7:1];
             data_i[7] <= rxd;
         end
      end

    assign receive_ack = (cur_st== RECEIVE_END)?1:0; //接收完成时，回复信号
endmodule
```

（4）时钟分频程序如下。

```
module clk_div(
  input clk,
  output reg clk_out
  );
localparam Baud_Rate=9600;
localparam div_num='d100000000/Baud_Rate;
reg [15:0] num;

always @ (posedge clk)
 begin
  if(num == div_num)
    begin
     num<=0;
     clk_out<=1;
    end
  else
   begin
    num<=num+1;
    clk_out<=0;
   end
 end

endmodule
```

五、实验结果

（1）编译编写好的程序，编译通过后，分配引脚，然后再次编译相关程序，生成下载文件。

（2）将程序下载到实验板上，连接实验板上的蓝牙串口，根据蓝牙芯片型号设置其模式。本实验中，蓝牙配置部分的 sw_pin[0]、sw_pin[1]为低电平，sw_pin[2]、sw_pin[3]、sw_pin[4]为高电平。连接蓝牙串口后，可以看到发送数据的回环结果。