第三讲 基于 Verilog HDL 和 Quartus Prime17.1 的电路设计与实现

学习目标：

通过下面几个实验，掌握基于 Verilog HDL 和 Quartus Prime17.1 的电路设计与实现。

① 3/8 译码器 => 扩展成 4/16 译码器
② 4 位二进制计数器 => 用 LED 显示
③ 流水灯（组合上面两个模块）
④ 流水灯（移位寄存器流水灯）
⑤ 组合上面两种流水灯（增加 2 选 1 模块）

1. 3/8 译码器扩展成 4/16 译码器

1.1 3/8 译码器

回顾之前的 3/8 译码器模块，3/8 译码器模块如图 1.1 所示。

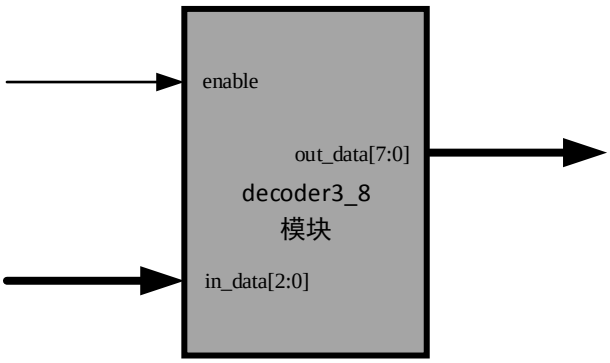

图 1.1 3/8 译码器模块

3/8 译码器的 Verilog 代码如下所示：

```
//文件名: decoder3_8.v
module  decoder3_8(
        input       [ 2: 0] in_data,    //3 位信号输入
        input               enable,     //使能信号输入
        output  reg [ 7: 0] out_data    //8 位译码信号输出
);

/* always @(*)语句的意思是 always 模块中的任何一个输入信号或电平发生变化时，
```

```
   该语句下方的模块将被执行,即所有输入信号都被视为敏感变量。
*/
//out_data: 3/8 译码器译码输出
always @(*)begin
    if(enable)begin
        case(in_data[2:0])  //enable = 1 时,对 in_data 进行译码，输出
out_data
            3'b000: out_data = 8'b0000_0001;
            3'b001: out_data = 8'b0000_0010;
            3'b010: out_data = 8'b0000_0100;
            3'b011: out_data = 8'b0000_1000;
            3'b100: out_data = 8'b0001_0000;
            3'b101: out_data = 8'b0010_0000;
            3'b110: out_data = 8'b0100_0000;
            3'b111: out_data = 8'b1000_0000;
            default:out_data = 8'b0000_0000;
        endcase
    end
    else begin  //enable = 0 时，输出恒为 8'b0000_0000
        out_data = 8'b0000_0000;
    end
end

endmodule
```

1.2 4/16 译码器

4/16 译码器真值表如表 1-1 所示。

表 1-1 4/16 译码器真值表

en	a[3]	a[2]	a[1]	a[0]	Y[15]	Y[14]	Y[13]	Y[12]	Y[11]	Y[10]	Y[9]	Y[8]	Y[7]	Y[6]	Y[5]	Y[4]	Y[3]	Y[2]	Y[1]	Y[0]
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
1	0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
1	1	0	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1

1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	x	x	x	x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

分析 4/16 译码器真值表可以发现：

当使能信号 en 为 1 且 a[3] = 0 时，对于相同的输入 a[2:0]，4/16 译码器的输出 Y[7:0]跟 3/8 译码器输出 Y[7:0]一致；

当使能信号 en 为 1 且 a[3] = 1 时，对于相同的输入 a[2:0]，4/16 译码器的输出 Y[15:8]跟 3/8 译码器输出 Y[7:0]一致；

当使能信号 en = 0 时，4/16 译码器输出 Y[15:0]全为 1。

这样，我们在编写 Verilog 代码时，在 4/16 译码器模块中调用两个 3/8 译码器模块；两个 3/8 译码器模块的 3 位输入信号为 4/16 译码器 4 位输入信号的低 3 位；4/16 译码器模块的 16 位输出由两个 3/8 译码器模块的 8 位输出组合得到；高 8 位译码输出的 3/8 译码器模块的使能信号由 4/16 译码器模块的使能信号和输入最高位相与得到；低 8 位译码输出的 3/8 译码器模块的使能信号由 4/16 译码器模块的使能信号和反向后的输入最高位相与得到。

4/16 译码器模块电路框图如图 1.2 所示。

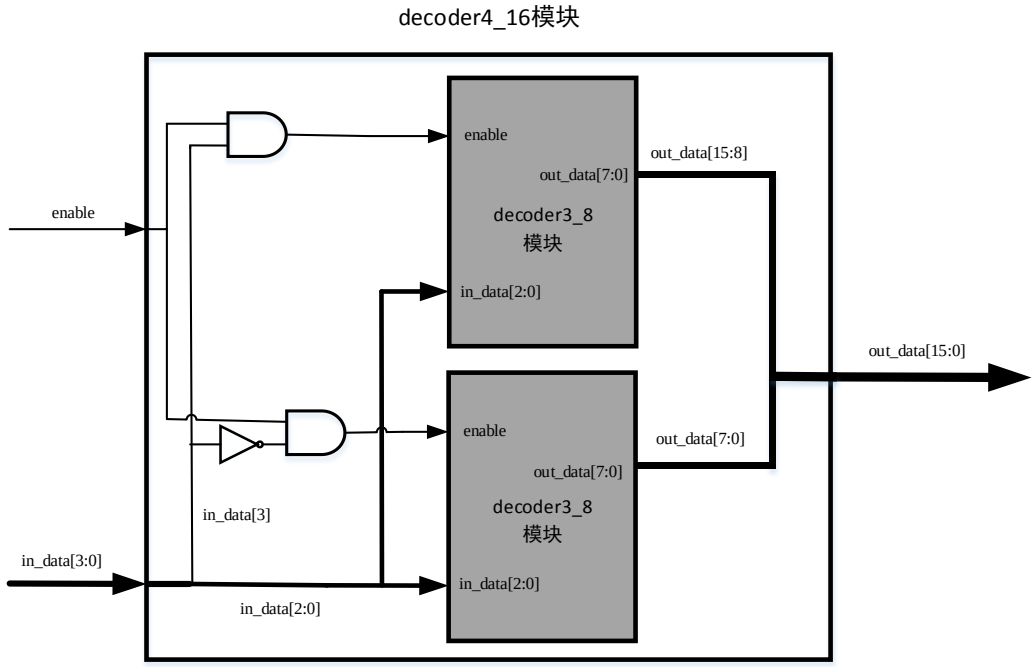

图 1.2 4/16 译码器模块电路框图

下面是 4/16 译码器的 Verilog 示例代码：

4/16 译码器模块有 4 位待译码输入信号、1 位使能信号输入和 16 位译码信号输出。

```
//文件名: decoder4_16.v
module  decoder4_16(
        input         [ 3: 0]   in_data,    //4 位信号输入
        input                   enable,     //使能信号输入
        output  wire  [15: 0]   out_data    //16 位译码信号输出
```

```
);

//定义两个 3/8 译码器的线型变量
wire enable0;
wire enable1;

//连续赋值语句
//当 enable = 1 且 a[3] = 0 时，enable0 = 1，否则为 0
//当 enable = 1 且 a[3] = 1 时，enable1 = 1，否则为 0
assign  enable0 =   enable && (~in_data[3]);
assign  enable1 =   enable && (in_data[3]);

//模块例化
//低八位
decoder3_8 decoder3_8_inst0(
    .in_data(in_data[2:0]),
    .enable(enable0),
    .out_data(out_data[7:0])
);

//高八位
decoder3_8 decoder3_8_inst1(
    .in_data(in_data[2:0]),
    .enable(enable1),
    .out_data(out_data[15:8])
);

endmodule
```

然后就可以按照第二讲的内容，使用 Quartus 软件对 4/16 译码器模块代码进行编译综合、布局布线，然后将编译生成的.sof 文件下载到我们的开发板中。

注意：需要添加 decoder3_8.v 文件。

可以自行选定要绑定的引脚，下面给出一个示例。

将 4/16 译码器模块的使能信号 enable 与 DE2-115 开发板中拨动开关 SW[4]绑定；

4 位待译码输入 in_data[3:0]与 DE2-115 开发板中拨动开关 SW[3:0]绑定；

将 16 位译码输出 out_data[15:0]与 DE2-115 开发板中 LEDR[15:0]绑定。

下面是引脚分配文件：

```
to              ,   location
enable          ,    PIN_AB27
in_data[3]      ,    PIN_AD27
in_data[2]      ,    PIN_AC27
in_data[1]      ,    PIN_AC28
in_data[0]      ,    PIN_AB28
```

```
out_data[15]    ,   PIN_G15
out_data[14]    ,   PIN_F15
out_data[13]    ,   PIN_H17
out_data[12]    ,   PIN_J16
out_data[11]    ,   PIN_H16
out_data[10]    ,   PIN_J15
out_data[9]     ,   PIN_G17
out_data[8]     ,   PIN_J17
out_data[7]     ,   PIN_H19
out_data[6]     ,   PIN_J19
out_data[5]     ,   PIN_E18
out_data[4]     ,   PIN_F18
out_data[3]     ,   PIN_F21
out_data[2]     ,   PIN_E19
out_data[1]     ,   PIN_F19
out_data[0]     ,   PIN_G19
```

使用 Quartus Prime17.1 软件设计开发请参照第二讲内容。

2. 4 位二进制计数器

本节需要设计一个 4 位二进制计数器，实现十进制 0~15 的自动循环计数。计数过程为：0-1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-0-1-2…依次循环，计数变化时间为 1 秒钟，通过一个拨动开关控制计数器停止和启动，并且在开发板上的 LED 上显示。

我们需要用到 DE2-115 开发板外部振荡器产生的 50MHz 时钟。

4 位二进制计数器模块设计思路如下:

① 通过外部 50MHz 时钟分频产生 1Hz 的时钟信号；

② 以 1Hz 的时钟信号作为敏感变量，编写时序逻辑代码，对 4 位二进制计数器进行计数。

下面是 4 位二进制计数器的 Verilog 示例代码：

4 位二进制计数器有一个时钟输入、一个低电平复位信号输入、计数器停止信号输入以及 4 位计数器输出。

```
//文件名：counter16.v
module  counter16(
       input             clk_50m,              //输入 50MHz 时钟
       input             rst_n,                //输入低电平复位信号
       input             cnt_stop,             //输入计数器停止信号
       output  reg [ 3: 0] cnt_led               //输出 4 位计数器输出
);

//变量声明
reg     [24:0]      cnt_50m                 ;   //50M 时钟计数，注意位宽为
25 位
reg               clk_1hz                   ;   //1Hz 时钟
```

```
//cnt_50m：50M时钟计数
//clk_50m上升沿作为敏感变量
always @(posedge clk_50m or negedge rst_n)begin
    if(!rst_n)begin
        cnt_50m <=  25'd0;
    end
    else if(cnt_50m == 25'd24_999_999)begin        //计数到
25'd24_999_999时复位到0
        cnt_50m <=  25'd0;
    end
    else begin
        cnt_50m <=  cnt_50m + 1'd1;               //否则计数加1
    end
end

//clk_1hz：1Hz时钟
always @(posedge clk_50m or negedge rst_n)begin
    if(!rst_n)begin
        clk_1hz <=  1'b0;
    end
    else if(cnt_50m == 25'd24999999)begin          //当cnt_50m ==
25'd24999999时，clk_1hz反向，产生1Hz的时钟信号
        clk_1hz <=  ~clk_1hz;
    end
end

//cnt_led：4位计数器输出
//clk_1hz作为敏感变量，计数每次变化时间为1秒
always @(posedge clk_1hz or negedge rst_n)begin
    if(!rst_n)begin
        cnt_led <=  4'd0;
    end
    else if(cnt_stop)begin
        cnt_led <= cnt_led;                       //当停止信号为1时，计数不
变
    end
    else if(cnt_led == 4'd15)begin                 //当cnt_led == 4'd15
时，复位到0
        cnt_led <=  4'd0;
    end
    else begin
        cnt_led <=  cnt_led + 1'd1;               //否则计数加1
    end
```

```
end

endmodule
```

然后就可以按照第二讲的内容，使用 Quartus 软件对 4 位二进制计数器模块代码进行编译综合、布局布线，然后将编译生成的.sof 文件下载到我们的开发板中。

可以自行选定要绑定的引脚，下面给出一个示例。

将 4 位二进制计数器的时钟 clk_50m 与 DE2-115 开发板中振荡器输出 clock_50（PIN_Y2）绑定；

将低电平复位信号 rst_n 与 DE2-115 开发板中的拨动开关 SW[17]绑定；

将计数器停止信号 cnt_stop 与 DE2-115 开发板中的拨动开关 SW[16]绑定；

将 4 位二进制计数输出 cnt_led[3:0]与 DE2-115 开发板中 LEDG[3:0]绑定。

下面是引脚分配文件：

```
to              ,   location
clk_50m         ,    PIN_Y2
rst_n           ,   PIN_Y23
cnt_stop        ,    PIN_Y24

cnt_led[3]      ,    PIN_E24
cnt_led[2]      ,    PIN_E25
cnt_led[1]      ,    PIN_E22
cnt_led[0]      ,    PIN_E21
```

使用 Quartus Prime17.1 软件设计开发请参照第二讲内容。

3. 计数器实现流水灯

本节需要组合第 1 节的 4/16 译码器模块和第 2 节的 4 位二进制计数器模块，完成一个以计数器实现的流水灯实验，通过一个拨动开关控制流水灯暂停和运行，并且在开发板上的 LED 上显示。

计数器实现流水灯模块的设计思路如下：

① 将外部 50MHz 时钟输入到 4 位二进制计数器模块中，产生自动循环 4 位二进制计数器；

② 将上述产生的自动循环 4 位二进制计数器与 4/16 译码器模块对应信号进行连接，输出 16 位译码信号；

③ 将外部低电平复位信号与 4 位二进制计数器模块的低电平复位信号和 4/16 译码器模块的使能信号连接；

④ 将外部流水灯暂停信号与 4 位二进制计数器模块的计数器停止信号连接。

下面是计数器实现流水灯的 Verilog 示例代码：

计数器实现流水灯模块有一个时钟输入、一个低电平复位信号输入、流水灯暂停信号输入以及 16 位流水灯信号输出。

```
//文件名：flow_led_counter16.v
module  flow_led_counter16(
      input                clk_50m,        //输入 50MHz 时钟
```

```
        input                   rst_n,          //输入低电平复位信号
        input                   flow_led_stop,  //输入流水灯暂停信号
        output  wire    [15: 0] flow_led        //输出 16 位流水灯信号
);

//定义线型变量
wire    [ 3: 0]     cnt_led;

//例化 4 位二进制计数器模块
counter16 counter16_inst(
    .clk_50m(clk_50m),                          //输入 50M 时钟
    .rst_n(rst_n),                              //低电平复位信号
    .cnt_stop(flow_led_stop),                   //输入计数器停止信号
    .cnt_led(cnt_led)                           //4 位计数器输出
);

//例化 4/16 译码器模块
decoder4_16 decoder4_16_inst(
    .in_data(cnt_led),                          //4 位信号输入
    .enable(rst_n),                             //使能信号输入
    .out_data(flow_led)                         //16 位译码信号输出
);

endmodule
```

然后就可以按照第二讲的内容，使用 Quartus 软件对计数器流水灯模块代码进行编译综合、布局布线，然后将编译生成的.sof 文件下载到我们的开发板中。

注意：需要添加 counter16.v、decoder4_16.v 和 decoder3_8.v 文件。

可以自行选定要绑定的引脚，下面给出一个示例。

将 4 位二进制计数器的时钟 clk_50m 与 DE2-115 开发板中振荡器输出 clock_50（PIN_Y2）绑定；

将低电平复位信号 rst_n 与 DE2-115 开发板中的拨动开关 SW[17]绑定；

将流水灯暂停信号 flow_led_stop 与 DE2-115 开发板中的拨动开关 SW[16]绑定；

将 16 位流水灯输出 flow_led [15:0]与 DE2-115 开发板中 LEDR[15:0]绑定。

下面是引脚分配文件：

```
to              ,   location
clk_50m         ,   PIN_Y2
rst_n           ,   PIN_Y23
flow_led_stop   ,   PIN_Y24

flow_led[15]    ,   PIN_G15
flow_led[14]    ,   PIN_F15
flow_led[13]    ,   PIN_H17
flow_led[12]    ,   PIN_J16
```

```
flow_led[11]    ,   PIN_H16
flow_led[10]    ,   PIN_J15
flow_led[9]     ,   PIN_G17
flow_led[8]     ,   PIN_J17
flow_led[7]     ,   PIN_H19
flow_led[6]     ,   PIN_J19
flow_led[5]     ,   PIN_E18
flow_led[4]     ,   PIN_F18
flow_led[3]     ,   PIN_F21
flow_led[2]     ,   PIN_E19
flow_led[1]     ,   PIN_F19
flow_led[0]     ,   PIN_G19
```

使用 Quartus Prime17.1 软件设计开发请参照第二讲内容。

4. 移位寄存器实现流水灯

本节需要通过移位寄存器实现流水灯，通过一个拨动开关控制流水灯暂停和运行，并且在开发板上的 LED 上显示。

移位寄存器实现实现流水灯模块的设计思路如下：

① 将外部 50MHz 时钟分频产生 1Hz 时钟信号；

② 以 1Hz 的时钟信号作为敏感变量，编写时序逻辑代码，通过左移移位寄存器（也可以使用右移移位寄存器）实现流水灯。

移位寄存器实现流水灯模块有一个时钟输入、一个低电平复位信号输入、流水灯暂停信号输入以及 16 位流水灯信号输出。

```
//文件名：flow_led_shift_register.v
module  flow_led_shift_register(
       input                clk_50m,          //输入 50MHz 系统时钟
       input                rst_n,            //输入低电平复位信号
       input                flow_led_stop,     //输入流水灯暂停信号
       output  reg [15: 0]    flow_led          //16 位流水灯输出信号
);

//变量声明
reg     [24:0]      cnt_50m;                    //50M 时钟计数，注意位宽为
25 位
reg               clk_1hz;                    //1Hz 时钟

//cnt_50m：50M 时钟计数
//clk_50m 上升沿作为敏感变量
always @(posedge clk_50m or negedge rst_n)begin
    if(!rst_n)begin
        cnt_50m <=  25'd0;
    end
```

```
    else if(cnt_50m == 25'd24_999_999)begin        //计数到
25'd24_999_999 时复位到 0
        cnt_50m <=  25'd0;
    end
    else begin
        cnt_50m <=  cnt_50m + 1'd1;               //否则计数加 1
    end
end

//clk_1hz: 1Hz 时钟
always @(posedge clk_50m or negedge rst_n)begin
    if(!rst_n)begin
        clk_1hz <=  1'b0;
    end
    else if(cnt_50m == 25'd24999999)begin          //当 cnt_50m ==
25'd24999999 时，clk_1hz 反向，产生 1Hz 的时钟信号
        clk_1hz <=  ~clk_1hz;
    end
end

//flow_led: 移位寄存器实现的 16 位流水灯
always @(posedge clk_1hz or negedge rst_n)begin
    if(!rst_n)begin
        flow_led <= 16'h0001;                     //flow_led 复位
    end
    else if(flow_led_stop)begin
        flow_led <= flow_led;                     //flow_led 保持不变，即流
水灯暂停
    end
    else begin
        flow_led <= {flow_led[14:0], flow_led[15]}; //左移移位寄存器实现
16 位流水灯
    end
end

endmodule
```

然后就可以按照第二讲的内容，使用 Quartus 软件对移位寄存器流水灯模块代码进行编译综合、布局布线，然后将编译生成的.sof 文件下载到我们的开发板中。

引脚绑定跟第 3 节计数器实现流水灯模块的引脚绑定一样。

使用 Quartus Prime17.1 软件设计开发请参照第二讲内容。

5. 组合流水灯

本节需要组合第 3 节计数器实现流水灯模块和第 4 节移位寄存器实现流水灯模块，增加一个 2 选 1 模块，通过一个拨动开关，改变流水灯的实现方式，并且通过另一个拨动开关控制流水灯暂停和运行，然后在开发板上的 LED 上显示。

组合流水灯模块的设计思路如下：

① 调用计数器实现流水灯模块，产生流水灯输出；

② 调用移位寄存器实现流水灯模块，产生流水灯输出；

③ 通过判断拨动开关状态，选择对应流水灯输出。

下面是组合流水灯的 Verilog 示例代码：

组合流水灯模块有一个时钟输入，一个低电平复位信号输入、流水灯暂停信号输入、流水灯选择信号输入以及 16 位流水灯信号输出。

```
//文件名：flow_led_combine.v
module  flow_led_combine(
       input                clk_50m,        //输入 50MHz 系统时钟
       input                rst_n,          //输入低电平复位信号
       input                flow_led_stop,  //输入流水灯暂停信号
       input                sw_change,      //2 选 1 选择信号
       output  wire    [15: 0] flow_led        //16 位流水灯输出信号
);

//定义线型变量
wire    [15: 0]     flow_led_cnt;             //计数器实现流水灯模块输出
wire    [15: 0]     flow_led_shift_reg;         //移位寄存器实现流水灯模块输出

//sw_change = 1 时，选择 flow_led_cnt 输出，否则选择 flow_led_shift_reg 输出
assign  flow_led = sw_change ? flow_led_cnt : flow_led_shift_reg;

//计数器实现流水灯模块例化
flow_led_counter16 flow_led_counter16_inst(
    .clk_50m(clk_50m),                      //输入 50MHz 系统时钟
    .rst_n(rst_n),                          //输入低电平复位信号
    .flow_led_stop(flow_led_stop),            //输入流水灯暂停信号
    .flow_led(flow_led_cnt)                   //输出 16 位流水灯
);

//移位寄存器实现流水灯模块例化
flow_led_shift_register(
    .clk_50m(clk_50m),                      //输入 50MHz 系统时钟
    .rst_n(rst_n),                          //输入低电平复位信号
    .flow_led_stop(flow_led_stop),            //输入流水灯暂停信号
    .flow_led(flow_led_shift_reg)             //16 位流水灯输出信号
);
```

```
endmodule
```

然后就可以按照第二讲的内容，使用 Quartus 软件对组合流水灯模块代码进行编译综合、布局布线，然后将编译生成的.sof 文件下载到我们的开发板中。

注意：需要添加 flow_led_counter16.v、flow_led_shift_register.v、counter16.v、decoder4_16.v 和 decoder3_8.v 文件。

可以自行选定要绑定的引脚，下面给出一个示例。

将 4 位二进制计数器的时钟 clk_50m 与 DE2-115 开发板中振荡器输出 clock_50（PIN_Y2）绑定；

将低电平复位信号 rst_n 与 DE2-115 开发板中的拨动开关 SW[17]绑定；

将流水灯暂停信号 flow_led_stop 与 DE2-115 开发板中的拨动开关 SW[16]绑定；

将流水灯选择信号 sw_change 与 DE2-115 开发板中的拨动开关 SW[15]绑定；

将 16 位流水灯输出 flow_led [15:0]与 DE2-115 开发板中 LEDR[15:0]绑定。

下面是引脚分配文件：

```
to              ,   location
clk_50m         ,    PIN_Y2
rst_n           ,   PIN_Y23
flow_led_stop   ,    PIN_Y24
sw_change       ,   PIN_AA22

flow_led[15]    ,    PIN_G15
flow_led[14]    ,    PIN_F15
flow_led[13]    ,    PIN_H17
flow_led[12]    ,    PIN_J16
flow_led[11]    ,    PIN_H16
flow_led[10]    ,    PIN_J15
flow_led[9]     ,    PIN_G17
flow_led[8]     ,    PIN_J17
flow_led[7]     ,    PIN_H19
flow_led[6]     ,    PIN_J19
flow_led[5]     ,    PIN_E18
flow_led[4]     ,    PIN_F18
flow_led[3]     ,    PIN_F21
flow_led[2]     ,    PIN_E19
flow_led[1]     ,    PIN_F19
flow_led[0]     ,    PIN_G19
```

使用 Quartus Prime17.1 软件设计开发请参照第二讲内容。

注意：由于两种方式实现的流水灯都是从右往左依次变化一位，所以有时候在拨动选择开关时，不能观察知道现在是哪种方式实现的流水灯。此时可以将移位寄存器实现流水灯模块复位赋值从 16’h0001 改成 16’h0003，这样流水灯每次会点亮两个，便于判别。