

课程名称：EDA 技术综合设计

设计报告名称：设计三 二进制转换十进制

班级：通信 214

姓名：王峤宇

学号：214022

一、设计内容及原理

基础任务

设计任务: 完成 4 位二进制整数转成十进制数, 输入由拨码开关给, 输出由数码管显示。

数码管设计: 利用视觉暂留, 采用动态扫描显示方式完成数码管设计, 每个数码管包括 7 个 LED 管和 1 个小圆点, 通过 8 个 IO 口控制一位数码管显示, 通过分时复用的扫描显示方案进行数码管驱动可以有效地提高对 IO 口资源的利用。板上提供的数码管为共阴极数码管。

二进制整数转成十进制: 读取拨码开关输入的二进制后, 通过逐位移位法的方式, 四位二进制数据转换成两个 4 位的 BCD 码保存。

提高内容

设计任务: 完成 4 位二进制纯小数转成十进制数, 输入由拨码开关给, 输出由数码管显示。

小数部分的设计较为困难, 但输入仅有 4 位二进制, 直接采用 case 语句完成所有情况的小数输出即可。

拓展任务

任务要求: 完成 8 位二进制实数转成十进制数 (其中四位整数, 四位小数), 输入由拨码开关给, 输出由数码管显示。

将前两部分任务设计的模块进行整合, 八位拨码开关, 分别作为四位整数输入和四位小数输入, 数码管两位用于整数输出, 四位用于小数部分输出。

二、设计过程

基础任务

通过逐位移位法实现的四位无符号数转化为 BCD 码的源文件如下:

Listing 1: 二进制转 BCD

```
1 module int2bcd_4bit(  
2     input  [3:0] uint4,  
3     output [3:0] outBCD0,  
4     output [3:0] outBCD1  
5 );  
6     reg [11:0] bcd_r0, bcd_r1, bcd_r2;  
7  
8     /* 直接移位两次 */  
9     always @(*) begin  
10         bcd_r0 = {6'b0, uint4, 2'b0};  
11     end
```

```

12
13  /* 移位第三次结果 */
14  always @(*) begin
15      bcd_r1 = {bcd_r0[10:0], 1'b0};
16      if (bcd_r1[7:4] > 4'b0100) bcd_r1[7:4] = bcd_r1[7:4] + 4'b0011;
17  end
18
19  /* 移位四次结果 */
20  always @(*) begin
21      bcd_r2 = {bcd_r1[10:0], 1'b0};
22  end
23
24  /* 输出结果 */
25  assign outBCD0 = bcd_r2[7:4];
26  assign outBCD1 = bcd_r2[11:8];
27 endmodule

```

时序扫描完成的数码管显示源文件如下, 以四个数码管为单位构建模块:

Listing 2: 数码管扫描显示

```

1 module scan_led_hex_disp_4(
2     input clk, rst,          /* 100MHz 时钟与复位 */
3     input [3:0] hex0, hex1, hex2, hex3, /* 显存 */
4     input [3:0] dp,          /* 小数点 */
5     output reg [3:0] an,      /* 片选信号 */
6     output reg [7:0] sseg     /* 八段数码管接口 */
7 );
8
9 localparam N = 16 + 2; /* 100MHz 时钟分频, 100Mhz/ 2^16 */
10 reg [N-1:0] regN;
11
12 always @(posedge clk, posedge rst) begin
13     if (rst) /* 异步, 高电平复位 */
14         regN <= 0;
15     else
16         regN <= regN + 1;
17 end
18
19 always @(*) begin /* 时分复用 */
20     case (regN[N-1:N-2])
21         2'b00: begin
22             an <= 4'b0001;
23             sseg[6:0] <= dt_translate(hex0);
24             sseg[7] <= dp[3];
25         end
26         2'b01: begin
27             an <= 4'b0010;
28             sseg[6:0] <= dt_translate(hex1);
29             sseg[7] <= dp[2];
30         end
31         2'b10: begin
32             an <= 4'b0100;

```

```

33         sseg[6:0] <= dt_translate(hex2);
34         sseg[7] <= dp[1];
35     end
36     2'b11: begin
37         an <= 4'b1000;
38         sseg[6:0] <= dt_translate(hex3);
39         sseg[7] <= dp[0];
40     end
41 endcase
42 end
43
44 function [6:0] dt_translate; /* 数码管译码函数 */
45     input [3:0] data;
46     begin
47         case(data)
48             4'd0: dt_translate = 7'b1111110; //number 0 -> 0x7e
49             4'd1: dt_translate = 7'b0110000; //number 1 -> 0x30
50             4'd2: dt_translate = 7'b1101101; //number 2 -> 0x6d
51             4'd3: dt_translate = 7'b1111001; //number 3 -> 0x79
52             4'd4: dt_translate = 7'b0110011; //number 4 -> 0x33
53             4'd5: dt_translate = 7'b1011011; //number 5 -> 0x5b
54             4'd6: dt_translate = 7'b1011111; //number 6 -> 0x5f
55             4'd7: dt_translate = 7'b1110000; //number 7 -> 0x70
56             4'd8: dt_translate = 7'b1111111; //number 8 -> 0x7f
57             4'd9: dt_translate = 7'b1111011; //number 9 -> 0x7b
58         endcase
59     end
60 endfunction
61 endmodule

```

整体系统的 top 文件如下:

Listing 3: top 文件

```

1 module bin2dec_top(
2     input clk, /* 100MHz */
3     input rst, /* 高电平异步复位 */
4     input [7:0] real_bin, /* 输入的四位整数加四位小数的二进制实数 */
5     output [7:0] sseg1, sseg2, /* 八段数码管 */
6     output [3:0] an1, an2 /* 片选信号 */
7 );
8
9 /* 计算整数部分的BCD码 */
10 wire [3:0] bcd_int0, bcd_int1;
11 int2bcd_4bit int2bcd_4bit_inst (
12     .uint4(real_bin[7:4]),
13     .outBCD0(bcd_int0),
14     .outBCD1(bcd_int1)
15 );
16
17 /* 第一个四数码管显示整数部分 */
18 scan_led_hex_disp_4 scan_led_hex_disp_4_inst_0 (
19     .clk(clk), .rst(rst),

```

```

20     .hex0(4'b0000), .hex1(4'b0000), .hex2(bcd_int1), .hex3(bcd_int0), .dp(4'b0001),
21     .an(an1), .sseg(sseg1)
22 );
23
24 scan_led_hex_disp_4 scan_led_hex_disp_4_inst_1 (
25     .clk(clk), .rst(rst),
26     .hex0(4'b0000), .hex1(4'b0000), .hex2(4'b0000), .hex3(4'b0000), .dp(4'b0000),
27     .an(an2), .sseg(sseg2)
28 );
29 endmodule

```

仿真文件如下:

Listing 4: 整数部分仿真文件

```

1 module bin2dec_top_tb;
2     reg clk;
3     reg rst;
4     reg [7:0] real_bin;
5     wire [7:0] sseg1, sseg2;
6     wire [3:0] an1, an2;
7
8     initial begin
9         clk = 0;
10        rst = 1;
11        #10;
12        clk = 1;
13        #10;
14        rst = 0;
15        real_bin = 8'b0;
16        forever begin
17            #100;
18            real_bin = real_bin + 8'b0001_0000;
19            if (real_bin == 8'b0) begin
20                $finish;
21            end
22        end
23    end
24    /* 生成时钟 */
25    always @(*) #5 clk <= ~clk;
26    /* 例化top */
27    bin2dec_top bin2dec_top_inst (
28        .clk(clk),
29        .rst(rst),
30        .real_bin(real_bin),
31        .sseg1(sseg1),
32        .sseg2(sseg2),
33        .an1(an1),
34        .an2(an2)
35    );
36 endmodule

```

提高任务

浮点数设计较为复杂,但由于输入位数较少,通过 case 语句实现,实现源文件如下:

Listing 5: 二进制小数转十进制

```
1 module float2bcd_4bit(  
2     input [3:0] float4,  
3     output [3:0] BCD0, BCD1, BCD2, BCD3  
4 );  
5     reg [3:0] BCD0_r, BCD1_r, BCD2_r, BCD3_r;  
6  
7     always @(*) begin  
8         case (float4)  
9             4'b0000: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = 16'b0; // 0.0000  
10            4'b0001: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd0, 4'd6, 4'd2, 4'd5}; // 0.0625  
11            4'b0010: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd1, 4'd2, 4'd5, 4'd0}; // 0.125  
12            4'b0011: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd1, 4'd8, 4'd7, 4'd5}; // 0.1875  
13            4'b0100: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd2, 4'd5, 4'd0, 4'd0}; // 0.25  
14            4'b0101: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd3, 4'd1, 4'd2, 4'd5}; // 0.3125  
15            4'b0110: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd3, 4'd7, 4'd5, 4'd0}; // 0.375  
16            4'b0111: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd4, 4'd3, 4'd7, 4'd5}; // 0.4375  
17            4'b1000: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd5, 4'd0, 4'd0, 4'd0}; // 0.5  
18            4'b1001: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd5, 4'd6, 4'd2, 4'd5}; // 0.5625  
19            4'b1010: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd6, 4'd2, 4'd5, 4'd0}; // 0.625  
20            4'b1011: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd6, 4'd8, 4'd7, 4'd5}; // 0.6875  
21            4'b1100: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd7, 4'd5, 4'd0, 4'd0}; // 0.75  
22            4'b1101: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd8, 4'd1, 4'd2, 4'd5}; // 0.8125  
23            4'b1110: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd8, 4'd7, 4'd5, 4'd0}; // 0.875  
24            4'b1111: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd9, 4'd3, 4'd7, 4'd5}; // 0.9375  
25            default: {BCD0_r, BCD1_r, BCD2_r, BCD3_r} = {4'd0, 4'd0, 4'd0, 4'd0}; // Default case  
26        endcase  
27    end  
28  
29    assign BCD0 = BCD0_r;  
30    assign BCD1 = BCD1_r;  
31    assign BCD2 = BCD2_r;  
32    assign BCD3 = BCD3_r;  
33 endmodule
```

修改后的 top 源文件新增内容如下:

Listing 6: 添加小数部分的 Top 文件修改内容

```
1 /* 计算小数部分的BCD码 */  
2 wire [3:0] bcd_float0, bcd_float1, bcd_float2, bcd_float3;  
3 float2bcd_4bit float2bcd_4bit_inst (  
4     .float4(real_bin[3:0]),  
5     .BCD0(bcd_float0), .BCD1(bcd_float1),  
6     .BCD2(bcd_float2), .BCD3(bcd_float3)  
7 );  
8 scan_led_hex_disp_4 scan_led_hex_disp_4_inst_1 (  
9     .clk(clk), .rst(rst),
```

```

10     .hex0(bcd_float0), .hex1(bcd_float1), .hex2(bcd_float2), .hex3(bcd_float3), .dp(4'b0000),
11     .an(an2), .sseg(sseg2)
12 );

```

对应的仿真文件的更改部分如下, 将激励改为遍历小数部分测试, 并且不再对数码管输出进行仿真测试, 直接测试小数输出 bcd 是否正确:

Listing 7: 小数转十进制仿真文件

```

1  initial begin
2      clk = 0;
3      rst = 1;
4      #10;
5      clk = 1;
6      #10;
7      rst = 0;
8      real_bin = 8'b0;
9      forever begin
10         #100;
11         real_bin = real_bin + 8'b0000_0001; /* 遍历浮点数情况 */
12         if (real_bin[3:0] == 4'b0) begin
13             $finish;
14         end
15     end
16 end

```

拓展任务

模块整合后的源文件如下:

Listing 8: 最终 top 源文件

```

1  module bin2dec_top(
2      input clk,                /* 100MHz */
3      input rst,                /* 高电平异步复位 */
4      input [7:0] real_bin,     /* 输入的四位整数加四位小数的二进制实数 */
5      output [7:0] sseg1, sseg2, /* 八段数码管 */
6      output [3:0] an1, an2     /* 片选信号 */
7  );
8
9      /* 计算整数部分的BCD码 */
10     wire [3:0] bcd_int0, bcd_int1;
11     int2bcd_4bit int2bcd_4bit_inst (
12         .uint4(real_bin[7:4]),
13         .outBCD0(bcd_int0),
14         .outBCD1(bcd_int1)
15     );
16
17     /* 第一个四数码管显示整数部分 */
18     scan_led_hex_disp_4 scan_led_hex_disp_4_inst_0 (
19         .clk(clk), .rst(rst),
20         .hex0(4'b0000), .hex1(4'b0000), .hex2(bcd_int1), .hex3(bcd_int0), .dp(4'b0001),

```

```

21     .an(an1), .sseg(sseg1)
22 );
23
24 /* 计算小数部分的BCD码 */
25 wire [3:0] bcd_float0, bcd_float1, bcd_float2, bcd_float3;
26 float2bcd_4bit float2bcd_4bit_inst (
27     .float4(real_bin[3:0]),
28     .BCD0(bcd_float0), .BCD1(bcd_float1),
29     .BCD2(bcd_float2), .BCD3(bcd_float3)
30 );
31 scan_led_hex_disp_4 scan_led_hex_disp_4_inst_1 (
32     .clk(clk), .rst(rst),
33     .hex0(bcd_float0), .hex1(bcd_float1), .hex2(bcd_float2), .hex3(bcd_float3), .dp(4'b0000),
34     .an(an2), .sseg(sseg2)
35 );
36 endmodule

```

仿真文件中的更改后的激励如下, 实现对所有可能输入数据的遍历:

Listing 9: 系统仿真激励

```

1  initial begin
2      clk = 0;
3      rst = 1;
4      #10;
5      clk = 1;
6      #10;
7      rst = 0;
8      real_bin = 8'b0;
9      forever begin
10         #100;
11         real_bin = real_bin + 8'b0000_0001; /* 遍历浮点数情况 */
12         if (real_bin == 8'b0) begin
13             $finish;
14         end
15     end
16 end

```

本系统设计最终需要用到, 八个数码管, 其中数码管为共阴极, 由三极管驱动, 整个系统的约束文件下:

Listing 10: 系统约束文件

```

1  set_property IOSTANDARD LVCMOS33 [get_ports clk]
2  set_property IOSTANDARD LVCMOS33 [get_ports rst]
3  set_property PACKAGE_PIN P17 [get_ports clk]
4  set_property PACKAGE_PIN R11 [get_ports rst]
5  set_property IOSTANDARD LVCMOS33 [get_ports {real_bin[7]}]
6  set_property IOSTANDARD LVCMOS33 [get_ports {real_bin[6]}]
7  set_property IOSTANDARD LVCMOS33 [get_ports {real_bin[5]}]
8  set_property IOSTANDARD LVCMOS33 [get_ports {real_bin[4]}]
9  set_property IOSTANDARD LVCMOS33 [get_ports {real_bin[3]}]
10 set_property IOSTANDARD LVCMOS33 [get_ports {real_bin[2]}]

```



```

11 set_property IOSTANDARD LVCMOS33 [get_ports {real_bin[1]}]
12 set_property IOSTANDARD LVCMOS33 [get_ports {real_bin[0]}]
13 set_property IOSTANDARD LVCMOS33 [get_ports {sseg1[7]}]
14 set_property IOSTANDARD LVCMOS33 [get_ports {sseg1[6]}]
15 set_property IOSTANDARD LVCMOS33 [get_ports {sseg1[5]}]
16 set_property IOSTANDARD LVCMOS33 [get_ports {sseg1[4]}]
17 set_property IOSTANDARD LVCMOS33 [get_ports {sseg1[3]}]
18 set_property IOSTANDARD LVCMOS33 [get_ports {sseg1[2]}]
19 set_property IOSTANDARD LVCMOS33 [get_ports {sseg1[1]}]
20 set_property IOSTANDARD LVCMOS33 [get_ports {sseg1[0]}]
21 set_property IOSTANDARD LVCMOS33 [get_ports {sseg2[7]}]
22 set_property IOSTANDARD LVCMOS33 [get_ports {sseg2[6]}]
23 set_property IOSTANDARD LVCMOS33 [get_ports {sseg2[5]}]
24 set_property IOSTANDARD LVCMOS33 [get_ports {sseg2[4]}]
25 set_property IOSTANDARD LVCMOS33 [get_ports {sseg2[3]}]
26 set_property IOSTANDARD LVCMOS33 [get_ports {sseg2[2]}]
27 set_property IOSTANDARD LVCMOS33 [get_ports {sseg2[1]}]
28 set_property IOSTANDARD LVCMOS33 [get_ports {sseg2[0]}]
29 set_property IOSTANDARD LVCMOS33 [get_ports {an1[3]}]
30 set_property IOSTANDARD LVCMOS33 [get_ports {an1[2]}]
31 set_property IOSTANDARD LVCMOS33 [get_ports {an1[1]}]
32 set_property IOSTANDARD LVCMOS33 [get_ports {an1[0]}]
33 set_property IOSTANDARD LVCMOS33 [get_ports {an2[3]}]
34 set_property IOSTANDARD LVCMOS33 [get_ports {an2[2]}]
35 set_property IOSTANDARD LVCMOS33 [get_ports {an2[1]}]
36 set_property IOSTANDARD LVCMOS33 [get_ports {an2[0]}]
37 set_property PACKAGE_PIN P5 [get_ports {real_bin[7]}]
38 set_property PACKAGE_PIN P4 [get_ports {real_bin[6]}]
39 set_property PACKAGE_PIN P3 [get_ports {real_bin[5]}]
40 set_property PACKAGE_PIN P2 [get_ports {real_bin[4]}]
41 set_property PACKAGE_PIN R2 [get_ports {real_bin[3]}]
42 set_property PACKAGE_PIN M4 [get_ports {real_bin[2]}]
43 set_property PACKAGE_PIN N4 [get_ports {real_bin[1]}]
44 set_property PACKAGE_PIN R1 [get_ports {real_bin[0]}]
45 set_property PACKAGE_PIN G2 [get_ports {an1[0]}]
46 set_property PACKAGE_PIN C2 [get_ports {an1[1]}]
47 set_property PACKAGE_PIN C1 [get_ports {an1[2]}]
48 set_property PACKAGE_PIN H1 [get_ports {an1[3]}]
49 set_property PACKAGE_PIN G6 [get_ports {an2[3]}]
50 set_property PACKAGE_PIN E1 [get_ports {an2[2]}]
51 set_property PACKAGE_PIN F1 [get_ports {an2[1]}]
52 set_property PACKAGE_PIN G1 [get_ports {an2[0]}]
53 set_property PACKAGE_PIN D5 [get_ports {sseg1[7]}]
54 set_property PACKAGE_PIN B4 [get_ports {sseg1[6]}]
55 set_property PACKAGE_PIN A4 [get_ports {sseg1[5]}]
56 set_property PACKAGE_PIN A3 [get_ports {sseg1[4]}]
57 set_property PACKAGE_PIN B1 [get_ports {sseg1[3]}]
58 set_property PACKAGE_PIN A1 [get_ports {sseg1[2]}]
59 set_property PACKAGE_PIN B3 [get_ports {sseg1[1]}]
60 set_property PACKAGE_PIN B2 [get_ports {sseg1[0]}]
61 set_property PACKAGE_PIN H2 [get_ports {sseg2[7]}]
62 set_property PACKAGE_PIN D4 [get_ports {sseg2[6]}]

```

```
63 set_property PACKAGE_PIN E3 [get_ports {sseg2[5]}]
64 set_property PACKAGE_PIN D3 [get_ports {sseg2[4]}]
65 set_property PACKAGE_PIN F4 [get_ports {sseg2[3]}]
66 set_property PACKAGE_PIN F3 [get_ports {sseg2[2]}]
67 set_property PACKAGE_PIN E2 [get_ports {sseg2[1]}]
68 set_property PACKAGE_PIN D2 [get_ports {sseg2[0]}]
```

三、仿真结果

基础任务

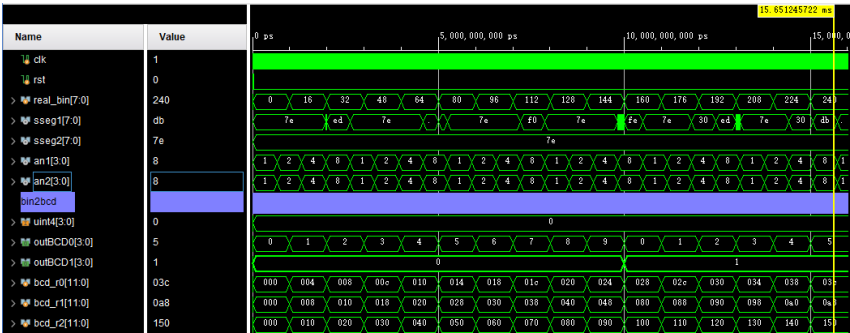


图 1: 整数部分仿真结果

对整数部分和数码管显示的仿真如图 1所示，仿真文件中控制每 1ms 控制输入的实数数据自增，整数部分从 0 自增到 15，相应的 BCD 码输出正确。显示部分扫描四个数码管的片选步进为 100MHz 的 216 分频，大概在 1.5KHz 左右，0.6ms。可以通过时序图看到随着片选信号的改变，数码管的输出也在随着改变。

提高任务

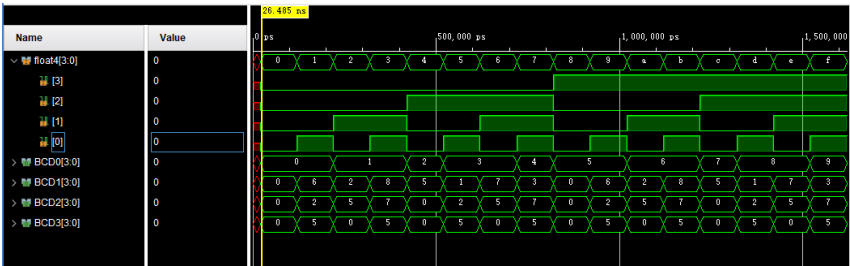


图 2: 小数部分仿真结果

对小数部分的仿真如图 1所示，仿真文件中控制每 100ns 控制输入的实数小数部分数据自增，小数部分的数据从 0000 到 1111，对应数据从 0000 到 0.9375，对应仿真图中数据为 16 进制显示，可以看到对应的 4 个 BCD 码从 0000 增加到 9375，对小数部分的 BCD 码解析输出无误。

拓展任务

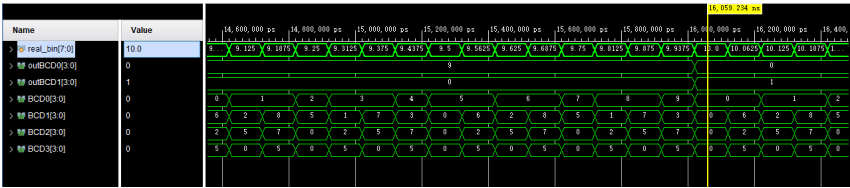


图 3: 实数设计仿真结果

对最终的实数设计的仿真如图 3所示, 仿真文件中控制每 100ns 控制输入的实数数据自增, 每次增加 0.0625, 实数的数据从 00000000 到 11111111, 对应数据从 0 到 15.9375, 对应仿真图中数据输入实数改为小数模式, 由于仿真窗口限制, 仅展示部分结果, 可以看到对应的十进制小数 (二进制存储状态) 被转化成六个 BCD 码, 并一一对应, 准确无误。

四、硬件验证结果

基础任务

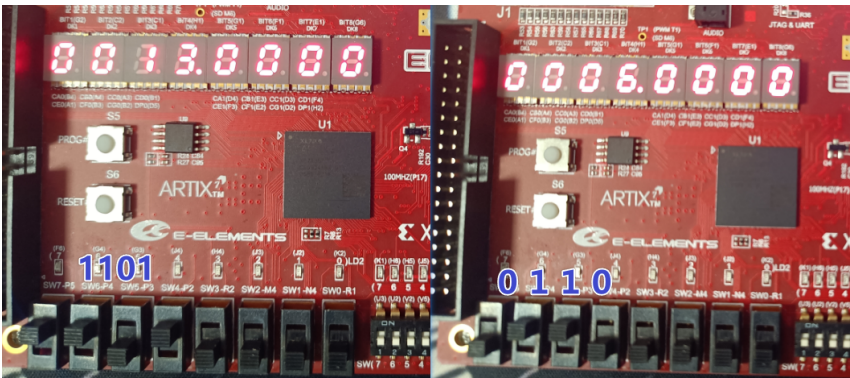


图 4: 整数部分硬件验证

针对整数部分的数码管的二进制转十进制硬件验证结果如图 4所示, 其中拨码开关从左开始数的高四位作为整数部分的输出, 由图中标注可以得到, 当输入为 4'b1101 时, 对应的十进制数是 13, 显示成功, 并且显示由小数点用于区分整数和小数。

提高任务

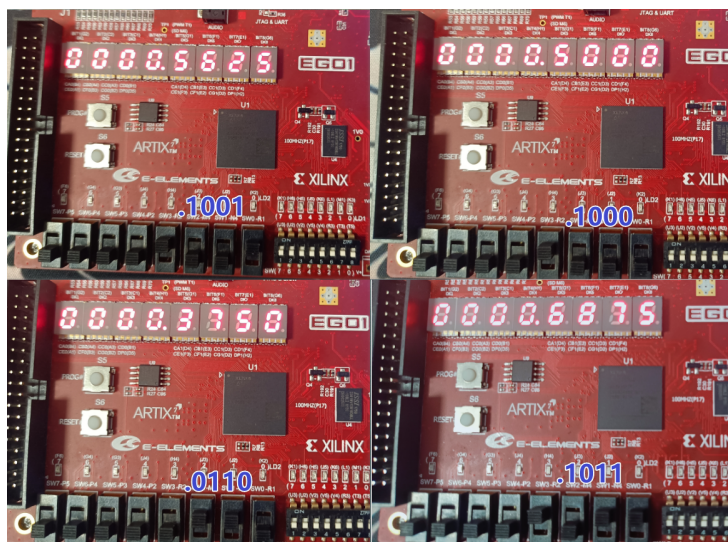


图 5: 小数部分硬件验证

针对小数部分的数码管的二进制转十进制硬件验证结果如图 5所示, 其中拨码开关从左开始数的后四个作为二进制小数部分的输入, 由图中标注可以得到, 当输入为.1001 时, 输出的十进制小数部分为.5625, .0110 时输出为.3750, 以.0110 为例, 输出为

$$.0110 : 0 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3} + 0 * 2^{-4} = 0.3750$$

拓展任务

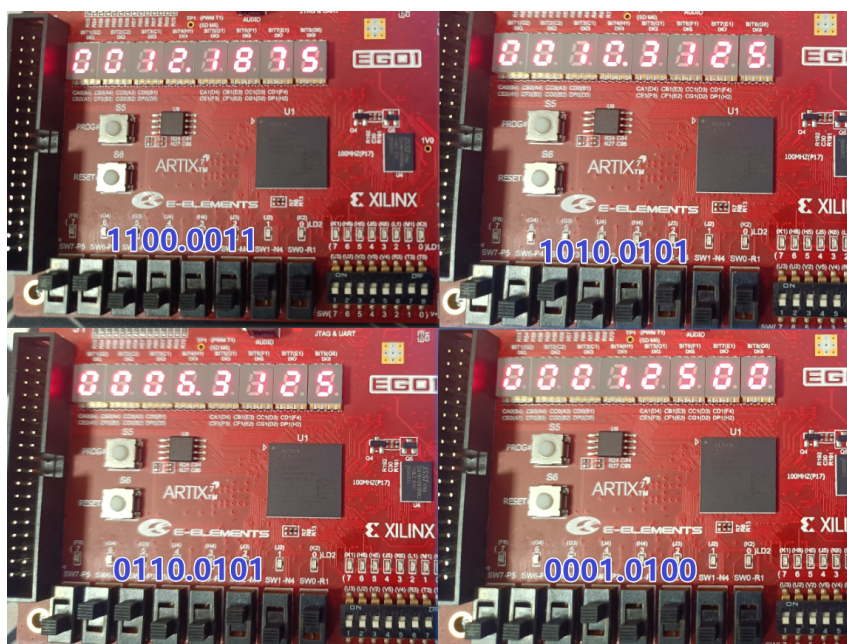


图 6: 八位实数硬件验证

针对小数部分的数码管的二进制转十进制硬件验证结果如图 6所示。

五、问题解决

设计初期, 对二进制转十进制的硬件实现思路不了解

解决: 通过查阅资料, 了解了二进制码转 BCD 码的逐位移位法, 并且在设计过程中对四位的逐位移位计算过程进行了一定的优化, 可以得到结论, 对于四位二进制转化为 BCD 码的过程中, 仅需要一次判断大四加三即可, 通过 wire 实现级间缓冲, 利用组合逻辑描述实现类似时序逻辑数据处理的效果。

六、写出心得体会

相比于实验二的简单门电路设计, 进行了一次稍微复杂, 模块更多的设计过程。为了使用数码管实现数据的输出, 提前接触了时序逻辑电路的设计, 包括时钟和分频的实现等。在进行对小数的仿真过程中, 偶然发现 Vivado 自带的仿真功能中, 不仅可以指定数据为无符号数, 十六进制等, 还可以将数据指定为复杂的浮点数显示形式, 便于对数据进行观察。再通过本次设计过程之后, 对 Vivado 和 Verilog HDL 更加熟料, 认识到硬件描述语言和计算机语言的区别, 逐渐养成边想电路边写代码的习惯。