课程名称：EDA 技术综合设计

设计报告名称：设计六 串行整数乘法器

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: 输入两个无符号四位二进制（BCD 码）整数，求出它们的乘法结果，输入由拨码开关给入，输入输出由数码管十进制显示。

二进制的整数乘法可以通过移位累加的方式实现。二进制整数乘法实现原理如图 1所示, 对于 BCD

```
        1 1 0 1  (13)
      x   1 0 1  (5)
 ---------------
        1 1 0 1
      0 0 0 0
    1 1 0 1
----------------
  1 0 0 0 0 0 1  (65)
```

图 1: 移位累加过程

码直接相乘, 可以直接视为二进制相乘。

提高内容

设计任务: 输入两个无符号四位二进制（8421 码）整数，求出它们的十进制乘法结果，输入由拨码开关给入，输入输出由数码管十进制显示。

拓展任务

任务要求: 输入两个无符号二进制整数（16-99 范围即可，也可更大但最大 256，这样输出用 5 位数码管，输入用 3 位数码管，刚好够用），求出它们的十进制乘法结果，输入由拨码开关给入，用数码管显示输入输出数值。

设计得到的用于测试整数乘法器的系统模块框图 2所示。

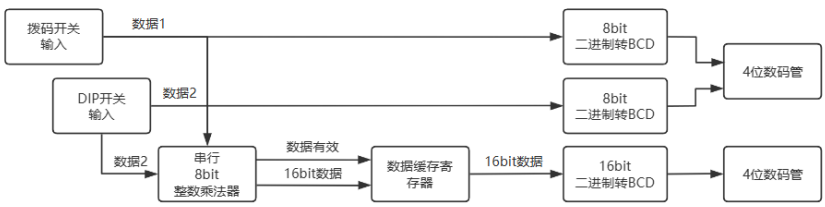

图 2: 系统模块框图

二、设计过程

基础任务

对于 BCD 码直接相乘的模块设计源文件如下:

Listing 1: 4 位 BCD 码相乘

```
/* 四位bcd乘法器 */
module mult_bcd(
    input clk,              /* 时钟 */
    input rst,              /* 异步高电平复位 */
    input data_in_rdy,      /* 输入数据有效 */
    input [3:0] bcd_in_1,   /* 输入乘数 */
    input [3:0] bcd_in_2,   /* 输入被乘数 */
    output data_out_rdy,    /* 输出数据有效 */
    output [3:0] bcd_out_1, /* 输出结果高位 */
    output [3:0] bcd_out_2  /* 输出结果低位 */
    );
    reg [1:0] cnt;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            cnt <= 2'b0;
        end
        else if (data_in_rdy) begin
            cnt <= cnt + 1'b1;
        end
        else begin
            cnt <= 2'b0;
        end
    end

    reg [7:0] num1;
    reg [3:0] num2;
    reg [7:0] acc;
    /* 累加求解乘法 */
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            acc <= 8'b0;
            num1 <= 8'b0;
            num2 <= 4'b0;
        end
        else if (data_in_rdy && cnt == 2'b0) begin
            num1 <= bcd_in_1 << 1;
            num2 <= bcd_in_2 >> 1;
            acc <= bcd_in_2[0] ? bcd_in_1 : 8'b0;
        end
        else if (cnt != 2'd3) begin
            num1 <= num1 << 1;
            num2 <= num2 >> 1;
            acc <= num2[0] ? num1 + acc : acc;
        end
```

```
45          else begin
46              acc <= acc;
47          end
48      end
49
50      reg [7:0] res;
51      reg data_out_rdy_r;
52      always @(posedge clk or posedge rst) begin
53          if (rst) begin
54              data_out_rdy_r <= 1'b0;
55              res <= 8'b0;
56          end
57          else if (cnt == 2'd3) begin
58              data_out_rdy_r <= 1'b1;
59              res <= acc;
60          end
61          else begin
62              data_out_rdy_r <= 1'b0;
63              res <= 8'b0;
64          end
65      end
66      assign data_out_rdy = data_out_rdy_r;
67
68      wire [3:0] bcd_foobar;
69      bin2bcd_8bit  bin2bcd_8bit_inst (
70          .bin(res),
71          .bcd_12bit({bcd_foobar, bcd_out_1, bcd_out_2})
72      );
73  endmodule
```

其中二进制转 BCD 码源文件如下:

Listing 2: 二进制转 BCD 码

```
1   module bin2bcd_8bit(
2       input [7:0] bin,
3       output [11:0] bcd_12bit  /* 高中低BCD结果 */
4       );
5
6       integer i;
7       reg [19:0] bcd_temp;
8       always @(*) begin
9           // 初始化, 直接移位三次
10          bcd_temp = {9'b0, bin, 3'b0};
11          // 逐位移位法
12          for (i = 0; i < 5; i = i + 1'b1) begin
13              // 如果BCD的每一部分 > 4, 加3
14              if (bcd_temp[11:8] > 4)
15                  bcd_temp[11:8] = bcd_temp[11:8] + 3;
16              if (bcd_temp[15:12] > 4)
17                  bcd_temp[15:12] = bcd_temp[15:12] + 3;
18              if (bcd_temp[19:16] > 4)
19                  bcd_temp[19:16] = bcd_temp[19:16] + 3;
```

```
            // 左移1位
            bcd_temp = {bcd_temp[18:0], 1'b0};
        end
    end
    assign bcd_12bit = bcd_temp[19-:12];
endmodule
```

top 文件如下:

Listing 3: top

```
module mult_bcd_top(
    input clk,                      /* 时钟 */
    input rst,                      /* 异步高电平复位 */
    input [7:0] keys,               /* 8个拨码开关输入 */
    input [7:0] dip,                /* dip开关输入 */
    output [7:0] sseg1, sseg2,      /* 八段数码管 */
    output [3:0] an1, an2           /* 数码管片选信号 */
    );

    /* 例化时分复用的乘法器模块 */
    wire [3:0] bcd_out_1;
    wire [3:0] bcd_out_2;
    wire data_out_rdy;
    mult_bcd  mult_bcd_inst (
        .clk(clk),
        .rst(rst),
        .data_in_rdy(1'b1),
        .bcd_in_1(keys[3:0]),
        .bcd_in_2(dip[3:0]),
        .data_out_rdy(data_out_rdy),
        .bcd_out_1(bcd_out_1),
        .bcd_out_2(bcd_out_2)
    );
    /* 第一个数码管显示 */
    reg [3:0] dsp_buffer_1[0:3];
    always @(posedge clk) begin
        dsp_buffer_1[0] <= keys[3:0];
        dsp_buffer_1[1] <= 4'd10;
        dsp_buffer_1[2] <= 4'd10;
        dsp_buffer_1[3] <= dip[3:0];
    end
    /* 第二个数码管显示 */
    reg [3:0] dsp_buffer_2[0:3];
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            dsp_buffer_2[0] <= 4'd10;
            dsp_buffer_2[1] <= 4'd10;
            dsp_buffer_2[2] <= 4'd10;
            dsp_buffer_2[3] <= 4'd10;
        end
        else if (data_out_rdy) begin
            dsp_buffer_2[0] <= 4'd10;
```

```
            dsp_buffer_2[1] <= 4'd10;
            dsp_buffer_2[2] <= bcd_out_1;
            dsp_buffer_2[3] <= bcd_out_2;
        end
        else begin
            dsp_buffer_2[0] <= 4'd10;
            dsp_buffer_2[1] <= 4'd10;
            dsp_buffer_2[2] <= dsp_buffer_2[2];
            dsp_buffer_2[3] <= dsp_buffer_2[3];
        end
    end
    /* 第一个四位数码管 */
    scan_led_hex_disp_4  scan_led_hex_disp_4_inst_1 (
        .clk(clk), .rst(rst),
        .hex0(dsp_buffer_1[0]), .hex1(dsp_buffer_1[1]),
        .hex2(dsp_buffer_1[2]), .hex3(dsp_buffer_1[3]),
        .dp(4'b0000), .an(an1), .sseg(sseg1)
    );
    /* 第二个四位数码管 */
    scan_led_hex_disp_4  scan_led_hex_disp_4_inst_2 (
        .clk(clk), .rst(rst),
        .hex0(dsp_buffer_2[0]), .hex1(dsp_buffer_2[1]),
        .hex2(dsp_buffer_2[2]), .hex3(dsp_buffer_2[3]),
        .dp(4'b0000), .an(an2), .sseg(sseg2)
    );
endmodule
```

针对模块的仿真文件如下:

Listing 4: BCD 乘法器仿真文件

```
module mult_bcd_tb;
    //Ports
    reg  clk;
    reg  rst;
    reg  data_in_rdy;
    reg [3:0] bcd_in_1;
    reg [3:0] bcd_in_2;
    wire  data_out_rdy;
    wire [3:0] bcd_out_1;
    wire [3:0] bcd_out_2;

    reg [3:0] bcd_res[0:1];
    initial begin
        clk = 0;
        rst = 0;
        #10;
        rst = 1;
        #10;
        rst = 0;
        bcd_in_1 = 4'd4;
        bcd_in_2 = 4'd5;
        data_in_rdy = 1'b1;
```

```
23
24          wait(data_out_rdy == 1'b1);
25          data_in_rdy = 1'b0;
26          bcd_res[0] = bcd_out_1;
27          bcd_res[1] = bcd_out_2;
28          #20;
29          $finish;
30      end
31
32      mult_bcd  mult_bcd_inst (
33          .clk(clk),
34          .rst(rst),
35          .data_in_rdy(data_in_rdy),
36          .bcd_in_1(bcd_in_1),
37          .bcd_in_2(bcd_in_2),
38          .data_out_rdy(data_out_rdy),
39          .bcd_out_1(bcd_out_1),
40          .bcd_out_2(bcd_out_2)
41      );
42
43      always #5  clk = ! clk ;
44  endmodule
```

基础任务、调高任务以及拓展任务公用的约束文件如下:

Listing 5: 约束文件

```
1  /* 100MHz时钟信号 */
2  set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports clk]
3  /* 复位信号 */
4  set_property -dict {PACKAGE_PIN R11 IOSTANDARD LVCMOS33} [get_ports rst]
5  /* 拨码开关 */
6  set_property -dict {PACKAGE_PIN P5 IOSTANDARD LVCMOS33} [get_ports {keys[7]}]
7  set_property -dict {PACKAGE_PIN P4 IOSTANDARD LVCMOS33} [get_ports {keys[6]}]
8  set_property -dict {PACKAGE_PIN P3 IOSTANDARD LVCMOS33} [get_ports {keys[5]}]
9  set_property -dict {PACKAGE_PIN P2 IOSTANDARD LVCMOS33} [get_ports {keys[4]}]
10 set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports {keys[3]}]
11 set_property -dict {PACKAGE_PIN M4 IOSTANDARD LVCMOS33} [get_ports {keys[2]}]
12 set_property -dict {PACKAGE_PIN N4 IOSTANDARD LVCMOS33} [get_ports {keys[1]}]
13 set_property -dict {PACKAGE_PIN R1 IOSTANDARD LVCMOS33} [get_ports {keys[0]}]
14 /* DIP */
15 set_property -dict {PACKAGE_PIN U3 IOSTANDARD LVCMOS33} [get_ports {dip[7]}]
16 set_property -dict {PACKAGE_PIN U2 IOSTANDARD LVCMOS33} [get_ports {dip[6]}]
17 set_property -dict {PACKAGE_PIN V2 IOSTANDARD LVCMOS33} [get_ports {dip[5]}]
18 set_property -dict {PACKAGE_PIN V5 IOSTANDARD LVCMOS33} [get_ports {dip[4]}]
19 set_property -dict {PACKAGE_PIN V4 IOSTANDARD LVCMOS33} [get_ports {dip[3]}]
20 set_property -dict {PACKAGE_PIN R3 IOSTANDARD LVCMOS33} [get_ports {dip[2]}]
21 set_property -dict {PACKAGE_PIN T3 IOSTANDARD LVCMOS33} [get_ports {dip[1]}]
22 set_property -dict {PACKAGE_PIN T5 IOSTANDARD LVCMOS33} [get_ports {dip[0]}]
23 /* 数码管 */
24 set_property -dict {PACKAGE_PIN G2 IOSTANDARD LVCMOS33} [get_ports {an1[0]}]
25 set_property -dict {PACKAGE_PIN C2 IOSTANDARD LVCMOS33} [get_ports {an1[1]}]
26 set_property -dict {PACKAGE_PIN C1 IOSTANDARD LVCMOS33} [get_ports {an1[2]}]
```

```
set_property -dict {PACKAGE_PIN H1 IOSTANDARD LVCMOS33} [get_ports {an1[3]}]
set_property -dict {PACKAGE_PIN G6 IOSTANDARD LVCMOS33} [get_ports {an2[3]}]
set_property -dict {PACKAGE_PIN E1 IOSTANDARD LVCMOS33} [get_ports {an2[2]}]
set_property -dict {PACKAGE_PIN F1 IOSTANDARD LVCMOS33} [get_ports {an2[1]}]
set_property -dict {PACKAGE_PIN G1 IOSTANDARD LVCMOS33} [get_ports {an2[0]}]
set_property -dict {PACKAGE_PIN D5 IOSTANDARD LVCMOS33} [get_ports {sseg1[7]}]
set_property -dict {PACKAGE_PIN B4 IOSTANDARD LVCMOS33} [get_ports {sseg1[6]}]
set_property -dict {PACKAGE_PIN A4 IOSTANDARD LVCMOS33} [get_ports {sseg1[5]}]
set_property -dict {PACKAGE_PIN A3 IOSTANDARD LVCMOS33} [get_ports {sseg1[4]}]
set_property -dict {PACKAGE_PIN B1 IOSTANDARD LVCMOS33} [get_ports {sseg1[3]}]
set_property -dict {PACKAGE_PIN A1 IOSTANDARD LVCMOS33} [get_ports {sseg1[2]}]
set_property -dict {PACKAGE_PIN B3 IOSTANDARD LVCMOS33} [get_ports {sseg1[1]}]
set_property -dict {PACKAGE_PIN B2 IOSTANDARD LVCMOS33} [get_ports {sseg1[0]}]
set_property -dict {PACKAGE_PIN H2 IOSTANDARD LVCMOS33} [get_ports {sseg2[7]}]
set_property -dict {PACKAGE_PIN D4 IOSTANDARD LVCMOS33} [get_ports {sseg2[6]}]
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports {sseg2[5]}]
set_property -dict {PACKAGE_PIN D3 IOSTANDARD LVCMOS33} [get_ports {sseg2[4]}]
set_property -dict {PACKAGE_PIN F4 IOSTANDARD LVCMOS33} [get_ports {sseg2[3]}]
set_property -dict {PACKAGE_PIN F3 IOSTANDARD LVCMOS33} [get_ports {sseg2[2]}]
set_property -dict {PACKAGE_PIN E2 IOSTANDARD LVCMOS33} [get_ports {sseg2[1]}]
set_property -dict {PACKAGE_PIN D2 IOSTANDARD LVCMOS33} [get_ports {sseg2[0]}]
```

提高任务

四位二进制乘法源码如下

Listing 6: 四位二进制乘法源码

```
module mult_bin(
    input clk,              /* 时钟 */
    input rst,              /* 异步高电平复位 */
    input data_in_rdy,      /* 输入数据有效 */
    input [7:0] data_in_1,  /* 输入乘数 */
    input [7:0] data_in_2,  /* 输入被乘数 */
    output data_out_rdy,    /* 输出数据有效 */
    output [15:0] data_out  /* 输出8位数据 */
    );
    reg [2:0] cnt;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            cnt <= 2'b0;
        end
        else if (data_in_rdy) begin
            cnt <= cnt + 1'b1;
        end
        else begin
            cnt <= 2'b0;
        end
    end

    reg [15:0] num1;
```

```
    reg [7:0] num2;
    reg [15:0] acc;
    /* 累加求解乘法 */
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            acc <= 16'b0;
            num1 <= 16'b0;
            num2 <= 8'b0;
        end
        else if (data_in_rdy && cnt == 2'b0) begin
            num1 <= data_in_1 << 1;
            num2 <= data_in_2 >> 1;
            acc <= data_in_2[0] ? data_in_1 : 16'b0;
        end
        else if (cnt != 3'd7) begin
            num1 <= num1 << 1;
            num2 <= num2 >> 1;
            acc <= num2[0] ? num1 + acc : acc;
        end
        else begin
            acc <= acc;
        end
    end

    reg [15:0] res;
    reg data_out_rdy_r;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            data_out_rdy_r <= 1'b0;
            res <= 16'b0;
        end
        else if (cnt == 3'd7) begin
            data_out_rdy_r <= 1'b1;
            res <= acc;
        end
        else begin
            data_out_rdy_r <= 1'b0;
            res <= 16'b0;
        end
    end
    assign data_out_rdy = data_out_rdy_r;
    assign data_out = res;
endmodule
```

top 文件如下:

Listing 7: top

```
module mult_bin_top(
    input clk,                      /* 时钟 */
    input rst,                      /* 异步高电平复位 */
    input [7:0] keys,               /* 8个拨码开关输入 */
    input [7:0] dip,                /* dip开关输入 */
```

```
    output [7:0] sseg1, sseg2,      /* 八段数码管 */
    output [3:0] an1, an2           /* 数码管片选信号 */
    );

    /* 例化时分复用的乘法器模块 */
    wire [15:0] res;
    wire data_out_rdy;
    mult_bin  mult_bin_inst (
        .clk(clk),
        .rst(rst),
        .data_in_rdy(1'b1),
        .data_in_1(keys),
        .data_in_2(dip),
        .data_out_rdy(data_out_rdy),
        .data_out(res)
    );

    /* 第一个数码管显示 */
    wire [3:0] dsp_buffer_1[0:3];
    bin2bcd_8bit  bin2bcd_8bit_inst_1 (
        .bin(keys),
        .bcd_12bit({dsp_buffer_1[0], dsp_buffer_1[1]})
    );
    bin2bcd_8bit  bin2bcd_8bit_inst_2 (
        .bin(dip),
        .bcd_12bit({dsp_buffer_1[2], dsp_buffer_1[3]})
    );

    /* 第二个数码管显示 */
    wire [3:0] dsp_buffer_2[0:3];
    bin2bcd_8bit  bin2bcd_8bit_inst (
        .bin(res),
        .bcd_12bit({dsp_buffer_2[1], dsp_buffer_2[2], dsp_buffer_2[3]})
    );

    reg [3:0] dsp_buffer_2_r[0:3];
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            dsp_buffer_2_r[0] <= 4'd0;
            dsp_buffer_2_r[1] <= 4'd0;
            dsp_buffer_2_r[2] <= 4'd0;
            dsp_buffer_2_r[3] <= 4'd0;
        end
        else if (data_out_rdy == 1'b1) begin
            dsp_buffer_2_r[0] <= dsp_buffer_2[0];
            dsp_buffer_2_r[1] <= dsp_buffer_2[1];
            dsp_buffer_2_r[2] <= dsp_buffer_2[2];
            dsp_buffer_2_r[3] <= dsp_buffer_2[3];
        end
        else begin
            dsp_buffer_2_r[0] <= dsp_buffer_2_r[0];
            dsp_buffer_2_r[1] <= dsp_buffer_2_r[1];
```

```
            dsp_buffer_2_r[2] <= dsp_buffer_2_r[2];
            dsp_buffer_2_r[3] <= dsp_buffer_2_r[3];
        end
    end

    /* 第一个四位数码管 */
    scan_led_hex_disp_4  scan_led_hex_disp_4_inst_1 (
        .clk(clk), .rst(rst),
        .hex0(dsp_buffer_1[0]), .hex1(dsp_buffer_1[1]),
        .hex2(dsp_buffer_1[2]), .hex3(dsp_buffer_1[3]),
        .dp(4'b0000), .an(an1), .sseg(sseg1)
    );
    /* 第二个四位数码管 */
    scan_led_hex_disp_4  scan_led_hex_disp_4_inst_2 (
        .clk(clk), .rst(rst),
        .hex0(dsp_buffer_2[0]), .hex1(dsp_buffer_2[1]),
        .hex2(dsp_buffer_2[2]), .hex3(dsp_buffer_2[3]),
        .dp(4'b0000), .an(an2), .sseg(sseg2)
    );
endmodule
```

仿真文件如下:

Listing 8: 仿真文件

```
module mult_bin_tb;
    reg  clk;
    reg  rst;
    reg  data_in_rdy;
    reg [7:0] data_in_1;
    reg [7:0] data_in_2;
    wire  data_out_rdy;
    wire [15:0] data_out;

    initial begin
        rst = 0;
        clk = 0;
        #10;
        rst = 1;
        #10;
        rst = 0;
        data_in_rdy = 1'b1;
        data_in_1 = 8'd15;
        data_in_2 = 8'd7;
        wait(data_out_rdy == 1'b1);
        #100;
        $finish;
    end

    mult_bin  mult_bin_inst (
        .clk(clk),
        .rst(rst),
        .data_in_rdy(data_in_rdy),
```

```
        .data_in_1(data_in_1),
        .data_in_2(data_in_2),
        .data_out_rdy(data_out_rdy),
        .data_out(data_out)
    );

    always #5  clk = ! clk ;
endmodule
```

拓展任务

源文件:

Listing 9: 8 位乘法器的源文件

```
module mult_bin(
    input clk,              /* 时钟 */
    input rst,              /* 异步高电平复位 */
    input data_in_rdy,      /* 输入数据有效 */
    input [7:0] data_in_1,  /* 输入乘数 */
    input [7:0] data_in_2,  /* 输入被乘数 */
    output data_out_rdy,    /* 输出数据有效 */
    output [15:0] data_out  /* 输出8位数据 */
    );
    reg [2:0] cnt;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            cnt <= 2'b0;
        end
        else if (data_in_rdy) begin
            cnt <= cnt + 1'b1;
        end
        else begin
            cnt <= 2'b0;
        end
    end

    reg [15:0] num1;
    reg [7:0] num2;
    reg [15:0] acc;
    /* 累加求解乘法 */
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            acc <= 16'b0;
            num1 <= 16'b0;
            num2 <= 8'b0;
        end
        else if (data_in_rdy && cnt == 2'b0) begin
            num1 <= data_in_1 << 1;
            num2 <= data_in_2 >> 1;
            acc <= data_in_2[0] ? data_in_1 : 16'b0;
```

```
        end
        else if (cnt != 3'd7) begin
            num1 <= num1 << 1;
            num2 <= num2 >> 1;
            acc <= num2[0] ? num1 + acc : acc;
        end
        else begin
            acc <= acc;
        end
    end

    reg [15:0] res;
    reg data_out_rdy_r;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            data_out_rdy_r <= 1'b0;
            res <= 16'b0;
        end
        else if (cnt == 3'd7) begin
            data_out_rdy_r <= 1'b1;
            res <= acc;
        end
        else begin
            data_out_rdy_r <= 1'b0;
            res <= 16'b0;
        end
    end
    assign data_out_rdy = data_out_rdy_r;
    assign data_out = res;
endmodule
```

16 位二进制转 BCD 码模块源文件:

Listing 10: 16bit 二进制转 BCD 码

```
module bin2bcd_16bit(
    input [15:0] bin,
    output [19:0] bcd_20bit  /* 高中低BCD结果 */
    );

    integer i;
    reg [35:0] bcd_temp;
    always @(*) begin
        // 初始化，直接移位三次
        bcd_temp = {17'b0, bin, 3'b0};
        // 逐位移位法
        for (i = 0; i < 13; i = i + 1'b1) begin
            // 如果BCD的每一部分 > 4，加3
            if (bcd_temp[19:16] > 4)
                bcd_temp[19:16] = bcd_temp[19:16] + 3;
            if (bcd_temp[23:20] > 4)
                bcd_temp[23:20] = bcd_temp[23:20] + 3;
            if (bcd_temp[27:24] > 4)
```

```
                bcd_temp[27:24] = bcd_temp[27:24] + 3;
            if (bcd_temp[31:28] > 4)
                bcd_temp[31:28] = bcd_temp[31:28] + 3;
            if (bcd_temp[35:32] > 4)
                bcd_temp[35:32] = bcd_temp[35:32] + 3;
            // 左移1位
            bcd_temp = {bcd_temp[34:0], 1'b0};
        end
    end
    assign bcd_20bit = bcd_temp[35-:20];
endmodule
```

top 文件:

Listing 11: 8 位乘法器 top 文件

```
module mult_bin_top(
    input clk,                      /* 时钟 */
    input rst,                      /* 异步高电平复位 */
    input [7:0] keys,               /* 8个拨码开关输入 */
    input [7:0] dip,                /* dip开关输入 */
    output [7:0] sseg1, sseg2,      /* 八段数码管 */
    output [3:0] an1, an2           /* 数码管片选信号 */
    );

    /* 例化时分复用的乘法器模块 */
    wire [15:0] res;
    wire data_out_rdy;
    mult_bin  mult_bin_inst (
        .clk(clk),
        .rst(rst),
        .data_in_rdy(1'b1),
        .data_in_1(keys),
        .data_in_2(dip),
        .data_out_rdy(data_out_rdy),
        .data_out(res)
    );

    /* 第一个数码管显示 */
    wire [3:0] dsp_buffer_1[0:3];
    bin2bcd_8bit  bin2bcd_8bit_inst_1 (
        .bin(keys),
        .bcd_12bit({dsp_buffer_1[0], dsp_buffer_1[1]})
    );
    bin2bcd_8bit  bin2bcd_8bit_inst_2 (
        .bin(dip),
        .bcd_12bit({dsp_buffer_1[2], dsp_buffer_1[3]})
    );

    /* 第二个数码管显示 */
    wire [3:0] dsp_buffer_2[0:3];
    bin2bcd_16bit  bin2bcd_16bit_inst (
        .bin(res),
```

```
        .bcd_20bit({dsp_buffer_2[0], dsp_buffer_2[1], dsp_buffer_2[2], dsp_buffer_2[3]})
    );

    reg [3:0] dsp_buffer_2_r[0:3];
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            dsp_buffer_2_r[0] <= 4'd0;
            dsp_buffer_2_r[1] <= 4'd0;
            dsp_buffer_2_r[2] <= 4'd0;
            dsp_buffer_2_r[3] <= 4'd0;
        end
        else if (data_out_rdy == 1'b1) begin
            dsp_buffer_2_r[0] <= dsp_buffer_2[0];
            dsp_buffer_2_r[1] <= dsp_buffer_2[1];
            dsp_buffer_2_r[2] <= dsp_buffer_2[2];
            dsp_buffer_2_r[3] <= dsp_buffer_2[3];
        end
        else begin
            dsp_buffer_2_r[0] <= dsp_buffer_2_r[0];
            dsp_buffer_2_r[1] <= dsp_buffer_2_r[1];
            dsp_buffer_2_r[2] <= dsp_buffer_2_r[2];
            dsp_buffer_2_r[3] <= dsp_buffer_2_r[3];
        end
    end

    /* 第一个四位数码管 */
    scan_led_hex_disp_4  scan_led_hex_disp_4_inst_1 (
        .clk(clk), .rst(rst),
        .hex0(dsp_buffer_1[0]), .hex1(dsp_buffer_1[1]),
        .hex2(dsp_buffer_1[2]), .hex3(dsp_buffer_1[3]),
        .dp(4'b0000), .an(an1), .sseg(sseg1)
    );
    /* 第二个四位数码管 */
    scan_led_hex_disp_4  scan_led_hex_disp_4_inst_2 (
        .clk(clk), .rst(rst),
        .hex0(dsp_buffer_2[0]), .hex1(dsp_buffer_2[1]),
        .hex2(dsp_buffer_2[2]), .hex3(dsp_buffer_2[3]),
        .dp(4'b0000), .an(an2), .sseg(sseg2)
    );
endmodule
```

仿真文件:

Listing 12: 8 位乘法器仿真文件

```
module mult_bin_tb;
    reg  clk;
    reg  rst;
    reg  data_in_rdy;
    reg [7:0] data_in_1;
    reg [7:0] data_in_2;
    wire  data_out_rdy;
    wire [15:0] data_out;
```

```

    initial begin
        rst = 0;
        clk = 0;
        #10;
        rst = 1;
        #10;
        rst = 0;
        data_in_rdy = 1'b1;
        data_in_1 = 8'd77;
        data_in_2 = 8'd33;
        wait(data_out_rdy == 1'b1);
        #100;
        $finish;
    end

    mult_bin  mult_bin_inst (
        .clk(clk),
        .rst(rst),
        .data_in_rdy(data_in_rdy),
        .data_in_1(data_in_1),
        .data_in_2(data_in_2),
        .data_out_rdy(data_out_rdy),
        .data_out(data_out)
    );

    always #5  clk = ! clk ;
endmodule
```

top 仿真文件:

Listing 13: top 仿真文件

```
module mult_bin_top_tb;

    // Parameters

    //Ports
    reg  clk;
    reg  rst;
    reg [7:0] keys;
    reg [7:0] dip;
    wire [7:0] sseg1;
    wire [7:0] sseg2;
    wire [3:0] an1;
    wire [3:0] an2;
    initial begin
        rst = 0;
        clk = 0;
        #10;
        rst = 1;
        #10;
        rst = 0;
```

```
21          keys = 8'd15;
22          dip = 8'd7;
23          #100;
24          $finish;
25      end
26
27      mult_bin_top  mult_bin_top_inst (
28          .clk(clk),
29          .rst(rst),
30          .keys(keys),
31          .dip(dip),
32          .sseg1(sseg1),
33          .sseg2(sseg2),
34          .an1(an1),
35          .an2(an2)
36      );
37
38      always #5  clk = ! clk ;
39
40  endmodule
```

三、仿真结果

基础任务

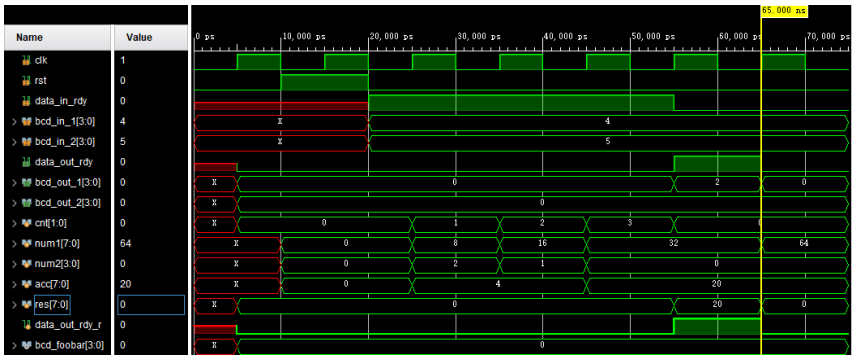

图 3: bcd 乘法器的仿真结果

针对 BCD 乘法器的仿真结果如图 3所示, 可以看到随着输入周期的变化, 系统按照时分复用的流程进行计算, 当指定输入数据有效之后, 经过四个周期完成运算, 得到 BCD 码 4 和 5 的乘积结果位 20。

提高任务

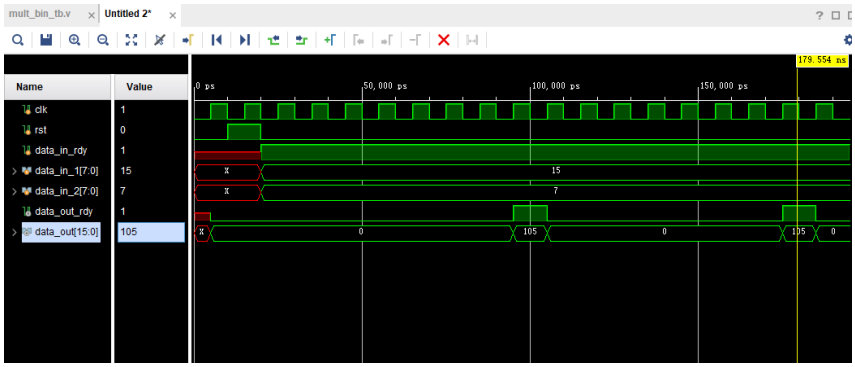

图 4: 四位二进制乘法的仿真结果

四位二进制乘法的验证结果如图 4所示, 其计算流程与 BCD 乘法器设计的工作流程一致, 计算结果正确无误。

拓展任务

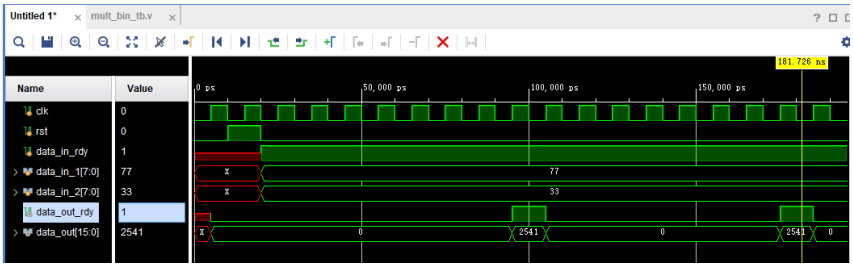

图 5: 八位二进制乘法的仿真结果

八位二进制乘法的验证结果如图 5所示, 通过 4 位二进制乘法改变位宽实现,8 个周期后完成计算, 计算结果 77*33=2541。

四、硬件验证结果

基础任务

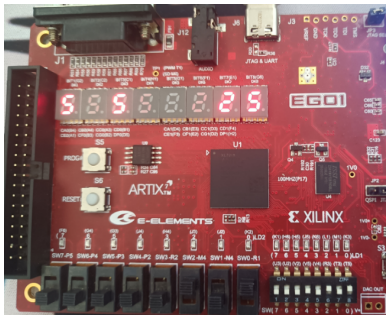

图 6: 5*5

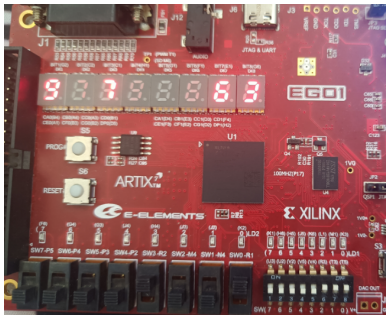

图 7: 9*7

硬件验证结果如图 6和图 7所示, 计算结果正确。

提高任务

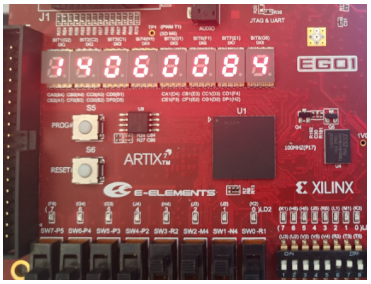

图 8: 14*6 = 84

四位二进制整数乘法的硬件验证结果如图 8, 计算 14*6 得到结果 84, 结果无误。

拓展任务

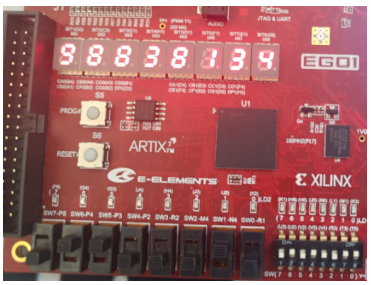

图 9: 98*83 = 8134

8 位二进制整数乘法的硬件验证结果如图 9, 计算 98*83 得到结果 8134, 结果无误。

五、问题解决

加载 bit 文件后, 数码管显示不正确

解决: 仿真时主要是对模块进行仿真, top 文件中完成的显示和输入没有进行直接的仿真, 导致未发现此次数码管问题, 烧录后数码管仅显示第一位, 起初以为是约束文件出错, 之后通过更改输入的方式发现输出一直, 得到结论: 问题出在数码管的部分。

首先通过 RTL 原理图检查问题, 可以直接发现是数码管例化时给的时钟错误, Vivado 下的 Verilog 语法检查器和综合器不会对未经声明而直接用于模块例化的信号产生 error, 仅仅会产生 warning, 导致此次错误发生。

构建 16bit 的 bin2bcd 模块, 不输出

解决: 图方便直接采用 8bit 的模块修改得到 16bit 的模块, 但是输出部分忘了修改, 新模块的输出是 bcd_20bit, 但没有修改原先的语句的左变量:

```
1 assign bcd_12bit = bcd_temp[35-:20];
```

但输出的 port 位 bcd_20bit, 导致错误, 通过查阅资料可以得知, Verilog 中支持未声明的线网类型数据, 如 wire, 直接使用 assign 语句声明的同时为其建立组合逻辑关系。所以综合器没有报错, 最终通过仿真层级关系一层一层检查, 最后发现问题。

修改代码后, 反复重新生成 bit 文件没有效果

修改代码后, 首先检查仿真结果, 仿真无误, 但是板上验证结果不对, 推测为 Vivado 在从综合开始的重新生成 bit 文件的过程中, 文件被缓存, 更改后的代码没有有效进行综合, 重启 vivado 后再次进行生成 bit 文件, 得到正确的结果。

六、写出心得体会

本次设计过程中通过 FPGA 完成了简单的算法设计, 也了解到 FPGA 中设计复杂算法的几种方法, 如果不采用合理的算法和设计结构将导致系统的资源占用过多, 甚至无法实现。基本的设计流程思路有并行和串行设计, 串行体现的是对资源的时分复用, 而并行就是构建好基础模块后, 重复模块实现复杂算法, 为了减少资源的占用, 通常选用串行设计思路。

为了解决串行计算中时分复用导致的效率低下, 通常还会采用流水线的设计方式, 将复杂的算法分为不同的处理阶段, 数据串行输入, 同一周期内不同数据进行不同的操作处理, 最终实现的效果就是一个周期可以得到一个计算结果, 在效率和资源上达到一定的平衡。