无03_王治_homework_数字逻辑与处理器基础实验_01

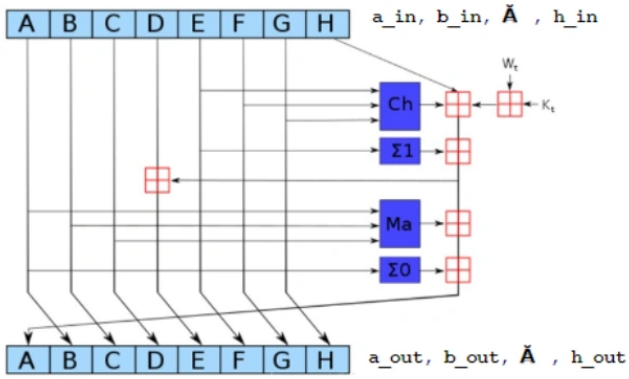
1.SHA-256 中的变换

SHA256 是 SHA-2 下细分出的一种密码散列函数算法，可以把消息或数据压缩成摘要。该函数将数据打乱混合，重新创建一个叫做散列值（或哈希值）的指纹。SHA256 广泛用于文件完整性检查、数字签名，以及某云盘的秒上传、比特币挖矿等功能。

其中，SHA256 中的一个核心为下面的映射（题 1 图）：

其中，SHA256 中的一个核心为下面的映射（题 1 图）：

```
S0 = (A rr 2) xor (A rr 13) xor (A rr 22)
t2 = S0 + Maj(A,B,C)
S1 = (E rr 6) xor (E rr 11) xor (E rr 25)
ch = (E and F) xor ((not E) and G)
t1 = H + S1 + ch + Kt + Wt
(A, B, C, D, E, F, G, H) = (t1+t2, A, B, C, D+t1, E, F, G)
```



题 1 图

其中

（1） 加法为 32 比特无符号加法，自然溢出，即结果为 $(A+B) \bmod 2^{32}$。

（2） Maj(A,B,C)为投票函数，A、B、C 三个输入中，如果对应比特中，有两个或三个 1，则 Maj(A,B,C)对应比特为 1，否则为 0。

例如：

```
A = 32'b10100000111100010010010111010101O;
B = 32'b00011000111110000110100001110010;
C = 32'b01000111010010111010100011000110;
Maj=32'b00000001111010100110100011100010;
```

（3） rr 为循环右移，移出的低位放到该数的高位

代码：

```Verilog
//Code_01_SHA-256中的变换
// round compression function
module sha256_round (
  input[31:0] Kt, Wt,
  input[31:0] a_in, b_in, c_in, d_in, e_in, f_in, g_in, h_in,
  output[31:0] a_out, b_out, c_out, d_out, e_out, f_out, g_out,h_out
  );
  // 请在此补充完整
  wire [31:0] S0,t2,t1,S1,ch,t1,Maj;
  sha256_S0 get_S0(a_in,S0);
  Maj get_Maj(a_in,b_in,c_in,Maj);
  assign t2=S0+Maj;
  sha256_S1 get_S1(e_in,S1);
  Ch get_ch(e_in,f_in,g_in,ch);
```

```
  assign t1=h_in+S1+ch+Kt+Wt;
  assign a_out=t1+t2;
  assign b_out=a_in;
  assign c_out=b_in;
  assign d_out=c_in;
  assign e_out=d_in+t1;
  assign f_out=e_in;
  assign g_out=f_in;
  assign h_out=g_in;
endmodule

// Σ0(x)
module sha256_S0 (
  input wire[31:0] x,
  output wire[31:0] S0
  );
  assign S0 =({x[1:0], x[31:2]}^{x[12:0], x[31:13]}^{x[21:0],x[31:22]});
endmodule

// Σ1(x)
module sha256_S1 (
  input wire[31:0] x,
  output wire[31:0] S1
  );
  // 请在此补充完整
  assign S1=({x[5:0],x[31:6]}^{x[10:0],x[31:11]}^{x[24:0],x[31:25]});
endmodule

// Ch(x,y,z)
module Ch (
  input wire[31:0] x, y, z,
  output wire[31:0] Ch
  );
  assign Ch =((x & y)^(~x & z));
endmodule

// Maj(x,y,z)
module Maj (
  input wire[31:0] x, y, z,
  output wire[31:0] Maj
  );
// 请在此补充完整
  assign Maj=(x&y)|(x&z)|(y&z);
endmodule
```

2.七段译码器的实现

要求：

要求：
(1) 将下面 BCD7 代码中的 assign 持续赋值语句换成 if-else 或 case 语句实现；
(2) 在 Vivado 中，将 top.v 中例化上面更改后的 BCD7.v，并综合实现，给出综合实现后的电路原理图结构；
(3) 找出控制七段数码管 d 段的 LUT，分析其配置字，手工验证其正确性。

1）代码

Verilog

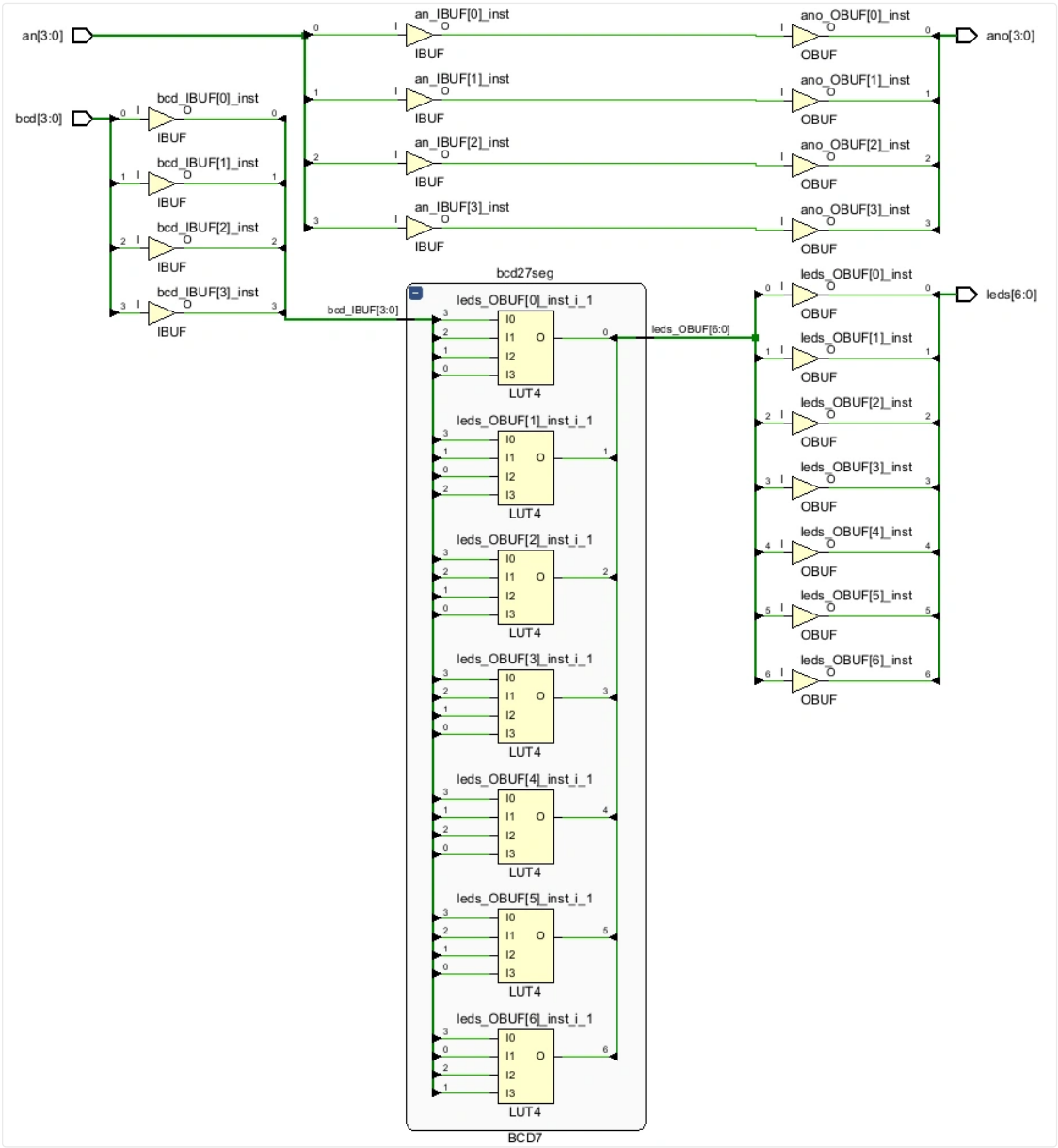
```
//Code_02_七段译码器的实现
module BCD7(
  din,dout
  );
```

```
  input[3:0]din;
  output[6:0]dout;

  //assign语句写法
  //wire[6:0] dout;
  //assign dout=(din==4'h0)?7'b0111111:
  //      (din==4'h1)?7'b0000110:
  //      (din==4'h2)?7'b1011011:
  //      (din==4'h3)?7'b1001111:
  //      (din==4'h4)?7'b1100110:
  //      (din==4'h5)?7'b1101101:
  //      (din==4'h6)?7'b1111101:
  //      (din==4'h7)?7'b0000111:
  //      (din==4'h8)?7'b1111111:
  //      (din==4'h9)?7'b1101111:7'b0;

  //case语句写法
  reg [6:0] dout;
  always @(*) begin
    case (din)
      4'h0: dout <= 7'b0111111;
      4'h1: dout <= 7'b0000110;
      4'h2: dout <= 7'b1011011;
      4'h3: dout <= 7'b1001111;
      4'h4: dout <= 7'b1100110;
      4'h5: dout <= 7'b1101101;
      4'h6: dout <= 7'b1111101;
      4'h7: dout <= 7'b0000111;
      4'h8: dout <= 7'b1111111;
      4'h9: dout <= 7'b1101111;
      default: dout <= 7'b0;
    endcase
  end
endmodule
```

2）原理图结构



3）LUT 配置字

控制七段数码管 d 段的 LUT 编号为 `leds_OBUF[3]_inst_i_1`，配置字为 `16'h1653`,转换为二进制为：`16'b0001 0110 0101 0011`。

分析可知：

当数字为：

- 0
- 2
- 3
- 5
- 6
- 8
- 9

时，七段数码管 d 段亮。

数字为：

- 1
- 4

- 7

时，七段数码管 d 段不亮。

而且 bcd_IBUF[3:0]接入 LUT 时，第 3/2/1/0 位对应的管脚为 I0/I1/I2/I3,所以要将输入数字的二进制反序。

配置字如图：

```
//十六进制16'h1653对应二进制16'hb0001_0110_0101_0011
//故：
0    1
1    1
2    0
3    0
4    1
5    0
6    1
7    0
8    0
9    1
10   1
11   0
12   1
13   0
14   0
15   0
```

输出数字	对应二进制	反序二进制	反序对应数字	配置字对应位置	d 段二极管亮灭
0	4'b0000	4'b0000	0	1	亮
1	4'b0001	4'b1000	8	0	灭
2	4'b0010	4'b0100	4	1	亮
3	4'b0011	4'b1100	12	1	亮
4	4'b0100	4'b0010	2	0	灭
5	4'b0101	4'b1010	10	1	亮
6	4'b0110	4'b0110	6	1	亮
7	4'b0111	4'b1110	14	0	灭
8	4'b1000	4'b0001	1	1	亮
9	4'b1001	4'b1001	9	1	亮