# 数字逻辑与处理器基础实验
# 第一次作业

无 81 马啸阳 2018011054

2020 年 4 月 5 日

## 1 实验目的

- 掌握基本组合逻辑的设计方法
  - 实现 SHA-256 中的变换
  - 实现七段译码器

## 2 实验原理

### 2.1 SHA-256

SHA-256 中的核心变换为如下的映射。
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)
其中 Maj(A,B,C) 为投票函数，rr 为循环右移。本实验中补全给出的各模块代码即可。

### 2.2 七段译码器

七段译码器通过将输入译码至共阳极的 LED 阴极输出，从而显示不同的数字。其逻辑框图如下所示。

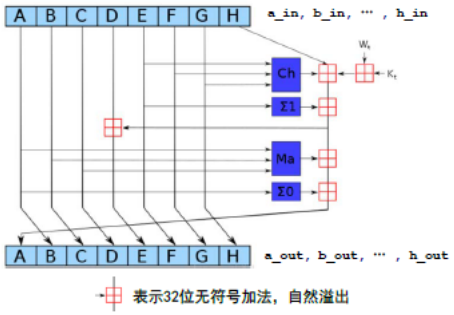


图 1: SHA-256 核心变换

本实验中，要将原七段译码器代码中的 assign 持续赋值语句换成 if-else 或 case 语句实现。为使用过程赋值语句，需要将原先的输出端口由 net 型变量转为 reg 型变量。同时本实验要求综合实现例化的七段译码器，并分析控制七段数码管的 LUT 配置字。

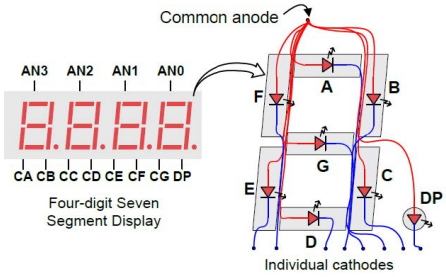


图 2: 七段 LED 显示器

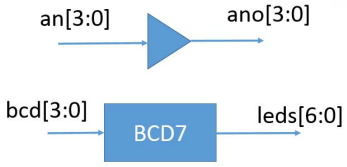


图 3: 七段译码器逻辑框图

# 3　实验代码及分析

## 3.1　文件清单

```
.
├── BCD7 ..... 七段译码管
│   ├── BCD7.v ..... 七段译码管模块
│   ├── top.v ..... 例化
│   └── top.xdc ..... 管脚约束
└── SHA256.v ..... SHA-256
```

## 3.2　SHA-256 中的变换

补全的 SHA-256 代码如下：

```
// 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
);
    wire [31:0] ch, maj_abc, s0, s1, t1, t2;
    Ch Ch0(.x(e_in), .y(f_in), .z(g_in), .Ch(ch));
    Maj Maj0(.x(a_in), .y(b_in), .z(c_in), .Maj(maj_abc));
    sha256_S0 s0_a(.x(a_in), .S0(s0));
    sha256_S1 s1_e(.x(e_in), .S1(s1));
    assign t1 = h_in + s1 + ch + Kt + Wt;
    assign t2 = s0 + maj_abc;
    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

// Σ (x)
```

```
25 module sha256_S0 (
26     input wire [31:0] x,
27     output wire [31:0] S0
28 );
29     assign S0 = ({x[1:0], x[31:2]} ^ {x[12:0], x[31:13]} ^ {x[21:0], x[31:22]});
30 endmodule
31 
32 // Σ (x)
33 module sha256_S1 (
34     input wire [31:0] x,
35     output wire [31:0] S1
36 );
37     assign S1 = ({x[5:0], x[31:6]} ^ {x[10:0], x[31:11]} ^ {x[24:0], x[31:25]});
38 endmodule
39 
40 // Ch(x,y,z)
41 module Ch (
42     input wire [31:0] x, y, z,
43     output wire [31:0] Ch
44 );
45     assign Ch = ((x & y) ^ (~x & z));
46 endmodule
47 
48 // Maj(x,y,z)
49 module Maj (
50     input wire [31:0] x, y, z,
51     output wire [31:0] Maj
52 );
53     assign Maj = ((x & y) ^ (y & z) ^ (z & x));
54 endmodule
```

## 3.3 七段译码器的实现

代码修改如下：

```
1 module BCD7(
2     din,
3     dout
4 );
```

```
5  input    [3:0]    din;
6  output   [6:0]    dout;
7
8  reg [6:0] dout;
9
10 always @(din)
11 begin
12     case (din)
13         4'h0: dout = 7'b1000000;
14         4'h1: dout = 7'b1111001;
15         4'h2: dout = 7'b0100100;
16         4'h3: dout = 7'b0110000;
17         4'h4: dout = 7'b0011001;
18         4'h5: dout = 7'b0010010;
19         4'h6: dout = 7'b0000010;
20         4'h7: dout = 7'b1111000;
21         4'h8: dout = 7'b0000000;
22         4'h9: dout = 7'b0010000;
23         default: dout = 7'b1111111;
24     endcase
25 end
26 endmodule
```

综合实现后的电路原理图结构及 LUT 对应表如下:

| LUT 单元 | 十六进制对应输出 | 二进制对应输出 |
|---|---|---|
| dout[0]_INST_0 | 16'hAA9C | 16'b1010_1010_1001_1100 |
| dout[1]_INST_0 | 16'hACE8 | 16'b1010_1100_1110_1000 |
| dout[2]_INST_0 | 16'hAAB0 | 16'b1010_1010_1011_0000 |
| dout[3]_INST_0 | 16'hEA9C | 16'b1110_1010_1001_1100 |
| dout[4]_INST_0 | 16'hFFB8 | 16'b1111_1111_1011_1000 |
| dout[5]_INST_0 | 16'hF9B8 | 16'b1111_1001_1011_1000 |
| dout[6]_INST_0 | 16'hEAA5 | 16'b1110_1010_1010_0101 |

表 1: LUT 对应表

以七段数码管 d 段为例, 其对应 dout[3], 对应输出为 16'b1110_1010_1001_1100。该 LUT

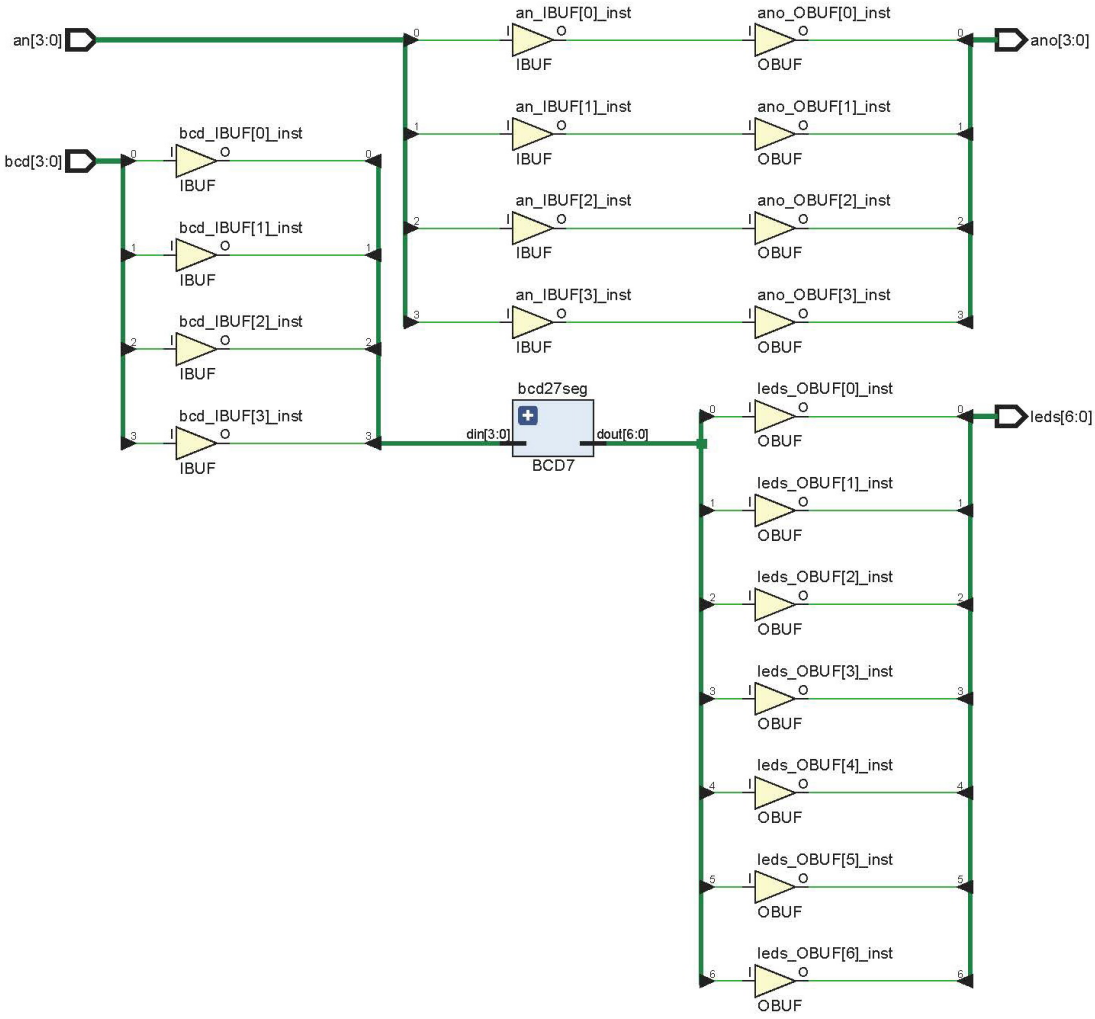
an[3:0]
bcd[3:0]
an_IBUF[0]_inst
an_IBUF[1]_inst
an_IBUF[2]_inst
an_IBUF[3]_inst
ano_OBUF[0]_inst
ano_OBUF[1]_inst
ano_OBUF[2]_inst
ano_OBUF[3]_inst
ano[3:0]
bcd_IBUF[0]_inst
bcd_IBUF[1]_inst
bcd_IBUF[2]_inst
bcd_IBUF[3]_inst
IBUF
OBUF
bcd27seg
din[3:0]
dout[6:0]
BCD7
leds_OBUF[0]_inst
leds_OBUF[1]_inst
leds_OBUF[2]_inst
leds_OBUF[3]_inst
leds_OBUF[4]_inst
leds_OBUF[5]_inst
leds_OBUF[6]_inst
leds[6:0]

图 4: 电路原理图

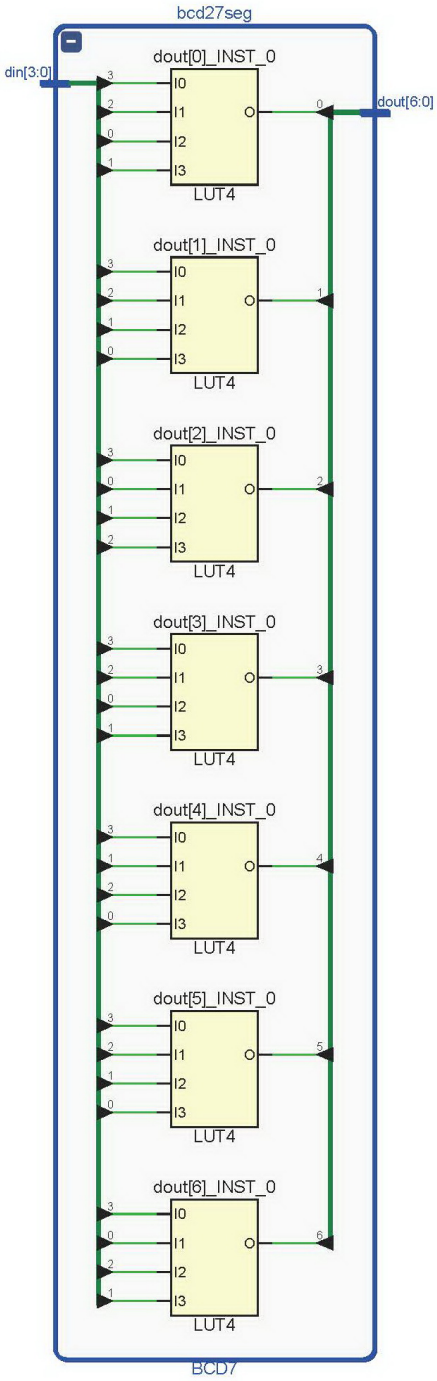
bcd27seg
din[3:0]
dout[6:0]
dout[0]_INST_0
dout[1]_INST_0
dout[2]_INST_0
dout[3]_INST_0
dout[4]_INST_0
dout[5]_INST_0
dout[6]_INST_0
I0
I1
I2
I3
O
LUT4
BCD7

图 5: BCD7

单元输入时，有对应 I0=bcd_IBUF[3]，I1=bcd_IBUF[2]，I2=bcd_IBUF[0]，I3=bcd_IBUF[1]。将 din，bcd_IBUF，I，dout[3] 的取值列表如下，验证符合七段数码管 d 段的控制取值，在 0、2、3、5、6、8、9 上亮。

| din | bcd_IBUF | I | dout[3] |
|---|---|---|---|
| 4'h0 | 4'b0000 | 4'b0000 | 0 |
| 4'h1 | 4'b0001 | 4'b0100 | 1 |
| 4'h2 | 4'b0010 | 4'b1000 | 0 |
| 4'h3 | 4'b0011 | 4'b1100 | 0 |
| 4'h4 | 4'b0100 | 4'b0010 | 1 |
| 4'h5 | 4'b0101 | 4'b0110 | 0 |
| 4'h6 | 4'b0110 | 4'b1010 | 0 |
| 4'h7 | 4'b0111 | 4'b1110 | 1 |
| 4'h8 | 4'b1000 | 4'b0001 | 0 |
| 4'h9 | 4'b1001 | 4'b0101 | 0 |
| 4'hA | 4'b1010 | 4'b1001 | 1 |
| 4'hB | 4'b1011 | 4'b1101 | 1 |
| 4'hC | 4'b1100 | 4'b0011 | 1 |
| 4'hD | 4'b1101 | 4'b0111 | 1 |
| 4'hE | 4'b1110 | 4'b1011 | 1 |
| 4'hF | 4'b1111 | 4'b1111 | 1 |

表 2: 七段数码管 d 段输出表

# 4　LUT 的输入端子

较多输入端子的 LUT 通常是由较少输入端子的 LUT 组合而成的。例如使用两个 4 输入 LUT，后一层加入一个二选一多路选择器则可实现一个 5 输入 LUT。一个 n 端口 LUT 可以实现 $2^{2^n}$ 种逻辑，因此要用 a 个 n 端口 LUT 实现实现一个 m 端口 LUT，需要满足 $\left(2^{2^n}\right)^a \geq 2^{2^m}$，即 $a \geq 2^{m-n}$。因此如果使用更少的输入端子，实现复杂的逻辑时，所需的 LUT 数目呈指数增长。

而如果 LUT 输入端子数目增多，单个 LUT 单元面积同上，也是指数增长的（功耗同时也一样指数增长）。因而芯片上所能放置的总 LUT 数目变少。同时，LUT 单元的延时也随输入端子数目增加而增加。单输入 LUT 单元（即二选一多路选择器）基本只有一级传输门延时，

而基本而言，延时随输入端子数目线性增长（每加一层可实现多一个输入端子）。使用简单逻辑时，更多输入 LUT 单元延时会更长，使电路性能下降。

综上所述，使用 4-6 个输入端子的 LUT 目前条件下比较经济。既不会因为输入端子过少而在实现复杂逻辑时需要大量单元影响可配置性，又不会因为输入端子过多而占用更大的面积，有较长延时和较大功耗。