数字逻辑与处理器基础实验

第一次作业

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）：

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) mod 232。
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’b10100001111000100100101110101010;

B = 32’b00011000111110000110100001110010;

C = 32’b01000111010010111010100011000110;

Maj=32’b00000001111010100110100011100010;

1. rr为循环右移，移出的低位放到该数的高位

**请根据上述信息，补全下面代码，实现上述功能。不要求使用Vivado进行综合仿真等。**

// 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

**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

**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

**endmodule**

1. 七段译码器的实现

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。

**module** BCD7**(**

din**,**

dout

**);**

**input** **[**3**:**0**]** din**;**

**output** **[**6**:**0**]** dout**;**

**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**;**

**endmodule**

1. 选做：LUT的输入端子

现代主流的商业FPGA产品中，所使用的的LUT的输入端的数目一般为4到6，比如本课程使用的Xilinx Artix-7系列的FPGA中就全部为6输入LUT。你是否赞同这种产品设计？如果同意，请分析LUT的输入端子数目更少或更大会有哪些代价；如果不同意，可给出更好的设计，并加以比较分析。提示：可从电路性能，占用芯片面积等角度分析。