高性能 SHA-256 电路设计规范

1. SHA(Security Hash Algorithm，安全散列算法)简介：

哈希函数可以根据消息的内容生成唯一的消息摘要（即，将输入的不定长数据转化为定长的数据摘要），从而验证消息的完整性。常见的哈希函数包括 SM3、MD4、MD5、SHA-1、SHA-256、SHA-384、SHA-512 等。由于哈希函数具有单向性和抗碰撞性，它被广泛用于口令的加密、消息认证码的构造、数字签名和伪随机数生成器等。

SHA 是由美国国家安全局设计，并由美国国家标准技术研究所发布为联邦数据处理标准的一系列标准哈希算法。SHA-256 是其中消息摘要位数为 256 位的一种算法。哈希算法通常作为硬件 IP 核集成于安全芯片中，其性能要求较高。

2. 课程设计目标：

面向哈希算法硬件实现的需求，设计一款高性能 SHA-256，完成电路的架构设计、Verilog HDL 代码设计、逻辑仿真、性能分析、逻辑综合、时序分析与验证和物理设计，进行结果分析比较。

3. 课程设计指标及应用要求：

3.1. SHA-256 电路设计要求：

（1） 对 SHA-256 算法进行循环展开，要求电路在一个时钟周期内完成两次哈希迭代，并对计算逻辑进行优化，减小关键路径的长度。
（2） 合理选择加法器的结构设计，尽可能提高 SHA-256 电路的性能。
（3） 要求电路能处理数据长度不大于 2^{12}=4096 比特的消息，根据 SHA-256 电路的吞吐率，合理选择所需的 IO 数量和输入缓存的容量。

注：消息填充（将不定长的消息填充为 512 比特的整数倍）的过程可认为由软件端完成，不需包含在电路中。

3.2. 芯片设计工艺：55nm 工艺；

3.3. SHA-256 电路的评价指标：

（1） 电路的性能，即时钟频率;
（2） 芯片的面积，包含 SHA-256 电路和芯片 IO。

4. SHA-256 算法介绍

SHA-256 算法的输入为不定长的消息，要求其长度小于2^{64}比特。首先，

SHA-256 算法需对输入消息按照一定规则进行填充，使其长度为 512 比特的整数倍。然后，再对各 512 比特数据块进行 64 次迭代运算，最终得到由 8 个 32 比特的数据组成的 256 比特消息摘要。

4.1. SHA-256 伪代码

Pseudo Algorithm: SHA-256	
1	**Function SHA-256(Message)**
2	//输入消息填充及解析，将消息长度填充为 N*512bits
3	**Message Padding and Parsing**
4	//对 N 个 512bits 数据块进行迭代计算
5	**FOR** i from 1 to N
6	//将 512bits 数据扩展为 64 个 32bits 数据
7	**FOR** j from 0 to 15
8	$W_j = M_j^{(i)}$ //$M^{(i)}$代表第 i 次迭代中的 512bits message
9	**END FOR**
10	**FOR** j from 16 to 63
11	$W_j = \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}$
12	**END FOR**
13	//HASH 值初始化
14	$a = H_0^{(i-1)};\ b = H_1^{(i-1)};\ c = H_2^{(i-1)};\ d = H_3^{(i-1)}$
15	$e = H_4^{(i-1)};\ f = H_5^{(i-1)};\ g = H_6^{(i-1)};\ h = H_7^{(i-1)}$
16	//对 HASH 值进行 64 次迭代计算，更新 ABCDEFGH
17	**FOR** j from 0 to 63
18	$T_1 \leftarrow h + \Sigma_1(e) + CH(e,f,g) + K_j + W_j$ //← 代表赋值操作
19	$T_2 \leftarrow \Sigma_0(a) + MAJ(a,b,c)$
20	$h \leftarrow g;\ g \leftarrow f;\ f \leftarrow e;\ e \leftarrow d + T_1$
21	$d \leftarrow c;\ c \leftarrow b;\ b \leftarrow a;\ a \leftarrow T_1 + T_2$
22	**END FOR**
23	//得到本轮 512-bit 数据块的 HASH 值
24	$H_0^{(i)} = a + H_0^{(i-1)};\ H_1^{(i)} = b + H_1^{(i-1)};\ H_2^{(i)} = c + H_2^{(i-1)};\ H_3^{(i)} = d + H_3^{(i-1)}$
25	$H_4^{(i)} = e + H_4^{(i-1)};\ H_5^{(i)} = f + H_5^{(i-1)};\ H_6^{(i)} = g + H_6^{(i-1)};\ H_7^{(i)} = h + H_7^{(i-1)}$
26	**END FOR**
27	**END Function**

4.2. 消息填充及解析(Message Padding and Parsing)

待处理的消息通常为字长不确定的数据，SHA-256 算法先将消息的长度填充为 512 比特的整数倍，再依次对 N 个 512 比特的数据进行 64 次迭代计算。填充过程如下所示：

Suppose a message has length L < 2^64. Before it is input to the hash function, the message is padded on the right as follows:

i. "1" is appended.
   Example: if the original message is "01010000", this is padded to "010100001".

ii. K "0"s are appended where K is the smallest, non-negative solution to the equation:

$$L + 1 + K = 448 \pmod{512}$$

iii. Then append the 64-bit block that is L in binary representation. After appending this block, the length of the message will be a multiple of 512 bits.

Example:

Suppose the original message is the bit string:

01100001 01100010 01100011 01100100 01100101

After step (a) this gives:

01100001 01100010 01100011 01100100 01100101 1

Since L = 40, the number of bits in the above is 41 and K = 407 "0"s are appended, making the total now 448. This gives the following in hex:

61626364 65800000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

The 64-bit representation of L = 40 is hex 00000000 00000028. Hence the final padded message is the following hex:

61626364 65800000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000028

4.3. 逻辑函数及算子

SHA-256 算法使用了 6 个逻辑函数，其中每个函数对 32bits 位宽的数据进行操作，每个函数的运算结果也为 32bits 位宽的数据。下式中，x, y, z 都为 32bits 位宽的数据。

$$CH(x,y,z) = (x\ AND\ y)\ XOR\ ((NOT\ x)\ AND\ z)$$

$$MAJ = (x\ AND\ y)\ XOR\ (x\ AND\ z)\ XOR\ (y\ AND\ z)$$

$$\Sigma_0(x) = ROTR\text{^}2(x)\ XOR\ ROTR\text{^}13(x)\ XOR\ ROTR\text{^}22(x)$$

$$\Sigma_1(x) = ROTR\text{^}6(x)\ XOR\ ROTR\text{^}11(x)\ XOR\ ROTR\text{^}25(x)$$

$$\sigma_0(x) = ROTR\text{^}7(x)\ XOR\ ROTR\text{^}18(x)\ XOR\ SHR\text{^}3(x)$$

$$\sigma_1(x) = ROTR\text{^}17(x)\ XOR\ ROTR\text{^}19(x)\ XOR\ SHR\text{^}10(x)$$

其中，AND/XOR/NOT 表示按位与/异或/非操作，ROTR 表示循环右移，SHR 表示逻辑右移。

$$X\ AND\ Y = bitwise\ logical\ \text{"and"}\ of\ X and\ Y$$

$X\ OR\ Y = bitwise\ logical\ "inclusive - \text{or}"\ of\ X and\ Y$

$X\ XOR\ Y = bitwise\ logical\ "exclusive - \text{or}"\ of\ X and\ Y$

$NOT\ X = bitwise\ logical\ "complement"\ of\ X$

$ROTR\hat{}n(x) = (x \gg n)\ OR\ \left(x \ll (w - n)\right)$

$SHR\hat{}n(x) = x \gg n$

4.4. 运算常量

SHA-256 的初始哈希值 H(0)，由以下八个 32bits 数据组成。它们是前八个质数的平方根的小数部分前 32 位。

$$H_0^{(0)} = 6a09e667; H_1^{(0)} = bb67ae85; H_2^{(0)} = 3c6ef372; H_3^{(0)} = a54ff53a$$

$$H_4^{(0)} = 510e527f; H_5^{(0)} = 9b05688c; H_6^{(0)} = 1f83d9ab; H_7^{(0)} = 5be0cd19$$

SHA-256 使用 64 个 32bits 运算常量 K0，K1，... ，K63。这些常量是前 64 个素数的立方根的小数部分前 32 位，用十六进制表示为：

K0~K7	428a2f98	71374491	b5c0fbcf	e9b5dba5	3956c25b	59f111f1	923f82a4	ab1c5ed5
K8~K15	d807aa98	12835b01	243185be	550c7dc3	72be5d74	80deb1fe	9bdc06a7	c19bf174
K16~K23	e49b69c1	efbe4786	0fc19dc6	240ca1cc	2de92c6f	4a7484aa	5cb0a9dc	76f988da
K24~K31	983e5152	a831c66d	b00327c8	bf597fc7	c6e00bf3	d5a79147	06ca6351	14292967
K32~K39	27b70a85	2e1b2138	4d2c6dfc	53380d13	650a7354	766a0abb	81c2c92e	92722c85
K40~K47	a2bfe8a1	a81a664b	c24b8b70	c76c51a3	d192e819	d6990624	f40e3585	106aa070
K48~K55	19a4c116	1e376c08	2748774c	34b0bcb5	391c0cb3	4ed8aa4a	5b9cca4f	682e6ff3
K56~K63	748f82ee	78a5636f	84c87814	8cc70208	90befffa	a4506ceb	bef9a3f7	c67178f2

5. 参考文献

[1] US Secure Hash Algorithms (SHA and HMAC-SHA)， https://www.rfc-editor.org/rfc/pdfrfc/rfc4634.txt.pdf

附录：SHA-256 运算示例

1. 原始输入消息（字符串）

abc

2. 填充后消息

0x61	0x62	0x63	0x80	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x18

3. 64 个扩展字

w[0]~w[3]:	0x61626380	0x00000000	0x00000000	0x00000000
w[4]~w[7]:	0x00000000	0x00000000	0x00000000	0x00000000
w[8]~w[11]:	0x00000000	0x00000000	0x00000000	0x00000000
w[12]~w[15]:	0x00000000	0x00000000	0x00000000	0x00000018
w[16]~w[19]:	0x61626380	0x000f0000	0x7da86405	0x600003c6
w[20]~w[23]:	0x3e9d7b78	0x0183fc00	0x12dcbfdb	0xe2e2c38e
w[24]~w[27]:	0xc8215c1a	0xb73679a2	0xe5bc3909	0x32663c5b
w[28]~w[31]:	0x9d209d67	0xec8726cb	0x702138a4	0xd3b7973b
w[32]~w[35]:	0x93f5997f	0x3b68ba73	0xaff4ffc1	0xf10a5c62
w[36]~w[39]:	0x0a8b3996	0x72af830a	0x9409e33e	0x24641522
w[40]~w[43]:	0x9f47bf94	0xf0a64f5a	0x3e246a79	0x27333ba3
w[44]~w[47]:	0x0c4763f2	0x840abf27	0x7a290d5d	0x065c43da
w[48]~w[51]	0xfb3e89cb	0xcc7617db	0xb9e66c34	0xa9993667
w[52]~w[55]:	0x84badedd	0xc21462bc	0x1487472c	0xb20f7a99
w[56]~w[59]:	0xef57b9cd	0xebe6b238	0x9fe3095e	0x78bc8d4b
w[60]~w[63]:	0xa43fcf15	0x668b2ff8	0xeeaba2cc	0x12b1edeb

4. 64 轮迭代（十六进制表示）

轮	a	b	c	d	e	f	g	h
0	6a09e667	bb67ae85	3c6ef372	a54ff53a	510e527f	9b05688c	1f83d9ab	5be0cd19
1	5d6aebcd	6a09e667	bb67ae85	3c6ef372	fa2a4622	510e527f	9b05688c	1f83d9ab
2	5a6ad9ad	5d6aebcd	6a09e667	bb67ae85	78ce7989	fa2a4622	510e527f	9b05688c
3	c8c347a7	5a6ad9ad	5d6aebcd	6a09e667	f92939eb	78ce7989	fa2a4622	510e527f
4	d550f666	c8c347a7	5a6ad9ad	5d6aebcd	24e00850	f92939eb	78ce7989	fa2a4622
5	04409a6a	d550f666	c8c347a7	5a6ad9ad	43ada245	24e00850	f92939eb	78ce7989
6	2b4209f5	04409a6a	d550f666	c8c347a7	714260ad	43ada245	24e00850	f92939eb
7	e5030380	2b4209f5	04409a6a	d550f666	9b27a401	714260ad	43ada245	24e00850
8	85a07b5f	e5030380	2b4209f5	04409a6a	0c657a79	9b27a401	714260ad	43ada245

9	8e04ecb9	85a07b5f	e5030380	2b4209f5	32ca2d8c	0c657a79	9b27a401	714260ad
10	8c87346b	8e04ecb9	85a07b5f	e5030380	1cc92596	32ca2d8c	0c657a79	9b27a401
11	4798a3f4	8c87346b	8e04ecb9	85a07b5f	436b23e8	1cc92596	32ca2d8c	0c657a79
12	f71fc5a9	4798a3f4	8c87346b	8e04ecb9	816fd6e9	436b23e8	1cc92596	32ca2d8c
13	87912990	f71fc5a9	4798a3f4	8c87346b	1e578218	816fd6e9	436b23e8	1cc92596
14	d932eb16	87912990	f71fc5a9	4798a3f4	745a48de	1e578218	816fd6e9	436b23e8
15	c0645fde	d932eb16	87912990	f71fc5a9	0b92f20c	745a48de	1e578218	816fd6e9
16	b0fa238e	c0645fde	d932eb16	87912990	07590dcd	0b92f20c	745a48de	1e578218
17	21da9a9b	b0fa238e	c0645fde	d932eb16	8034229c	07590dcd	0b92f20c	745a48de
18	c2fbd9d1	21da9a9b	b0fa238e	c0645fde	846ee454	8034229c	07590dcd	0b92f20c
19	fe777bbf	c2fbd9d1	21da9a9b	b0fa238e	cc899961	846ee454	8034229c	07590dcd
20	e1f20c33	fe777bbf	c2fbd9d1	21da9a9b	b0638179	cc899961	846ee454	8034229c
21	9dc68b63	e1f20c33	fe777bbf	c2fbd9d1	8ada8930	b0638179	cc899961	846ee454
22	c2606d6d	9dc68b63	e1f20c33	fe777bbf	e1257970	8ada8930	b0638179	cc899961
23	a7a3623f	c2606d6d	9dc68b63	e1f20c33	49f5114a	e1257970	8ada8930	b0638179
24	c5d53d8d	a7a3623f	c2606d6d	9dc68b63	aa47c347	49f5114a	e1257970	8ada8930
25	1c2c2838	c5d53d8d	a7a3623f	c2606d6d	2823ef91	aa47c347	49f5114a	e1257970
26	cde8037d	1c2c2838	c5d53d8d	a7a3623f	14383d8e	2823ef91	aa47c347	49f5114a
27	b62ec4bc	cde8037d	1c2c2838	c5d53d8d	c74c6516	14383d8e	2823ef91	aa47c347
28	77d37528	b62ec4bc	cde8037d	1c2c2838	edffbff8	c74c6516	14383d8e	2823ef91
29	363482c9	77d37528	b62ec4bc	cde8037d	6112a3b7	edffbff8	c74c6516	14383d8e
30	a0060b30	363482c9	77d37528	b62ec4bc	ade79437	6112a3b7	edffbff8	c74c6516
31	ea992a22	a0060b30	363482c9	77d37528	0109ab3a	ade79437	6112a3b7	edffbff8
32	73b33bf5	ea992a22	a0060b30	363482c9	ba591112	0109ab3a	ade79437	6112a3b7
33	98e12507	73b33bf5	ea992a22	a0060b30	9cd9f5f6	ba591112	0109ab3a	ade79437
34	fe604df5	98e12507	73b33bf5	ea992a22	59249dd3	9cd9f5f6	ba591112	0109ab3a
35	a9a7738c	fe604df5	98e12507	73b33bf5	085f3833	59249dd3	9cd9f5f6	ba591112
36	65a0cfe4	a9a7738c	fe604df5	98e12507	f4b002d6	085f3833	59249dd3	9cd9f5f6
37	41a65cb1	65a0cfe4	a9a7738c	fe604df5	0772a26b	f4b002d6	085f3833	59249dd3
38	34df1604	41a65cb1	65a0cfe4	a9a7738c	a507a53d	0772a26b	f4b002d6	085f3833
39	6dc57a8a	34df1604	41a65cb1	65a0cfe4	f0781bc8	a507a53d	0772a26b	f4b002d6
40	79ea687a	6dc57a8a	34df1604	41a65cb1	1efbc0a0	f0781bc8	a507a53d	0772a26b
41	d6670766	79ea687a	6dc57a8a	34df1604	26352d63	1efbc0a0	f0781bc8	a507a53d
42	df46652f	d6670766	79ea687a	6dc57a8a	838b2711	26352d63	1efbc0a0	f0781bc8
43	17aa0dfe	df46652f	d6670766	79ea687a	decd4715	838b2711	26352d63	1efbc0a0
44	9d4baf93	17aa0dfe	df46652f	d6670766	fda24c2e	decd4715	838b2711	26352d63
45	26628815	9d4baf93	17aa0dfe	df46652f	a80f11f0	fda24c2e	decd4715	838b2711
46	72ab4b91	26628815	9d4baf93	17aa0dfe	b7755da1	a80f11f0	fda24c2e	decd4715
47	a14c14b0	72ab4b91	26628815	9d4baf93	d57b94a9	b7755da1	a80f11f0	fda24c2e
48	4172328d	a14c14b0	72ab4b91	26628815	fecf0bc6	d57b94a9	b7755da1	a80f11f0
49	05757ceb	4172328d	a14c14b0	72ab4b91	bd714038	fecf0bc6	d57b94a9	b7755da1
50	f11bfaa8	05757ceb	4172328d	a14c14b0	6e5c390c	bd714038	fecf0bc6	d57b94a9
51	7a0508a1	f11bfaa8	05757ceb	4172328d	52f1ccf7	6e5c390c	bd714038	fecf0bc6

52	886e7a22	7a0508a1	f11bfaa8	05757ceb	49231c1e	52f1ccf7	6e5c390c	bd714038
53	101fd28f	886e7a22	7a0508a1	f11bfaa8	529e7d00	49231c1e	52f1ccf7	6e5c390c
54	f5702fdb	101fd28f	886e7a22	7a0508a1	9f4787c3	529e7d00	49231c1e	52f1ccf7
55	3ec45cdb	f5702fdb	101fd28f	886e7a22	e50e1b4f	9f4787c3	529e7d00	49231c1e
56	38cc9913	3ec45cdb	f5702fdb	101fd28f	54cb266b	e50e1b4f	9f4787c3	529e7d00
57	fcd1887b	38cc9913	3ec45cdb	f5702fdb	9b5e906c	54cb266b	e50e1b4f	9f4787c3
58	c062d46f	fcd1887b	38cc9913	3ec45cdb	7e44008e	9b5e906c	54cb266b	e50e1b4f
59	ffb70472	c062d46f	fcd1887b	38cc9913	6d83bfc6	7e44008e	9b5e906c	54cb266b
60	b6ae8fff	ffb70472	c062d46f	fcd1887b	b21bad3d	6d83bfc6	7e44008e	9b5e906c
61	b85e2ce9	b6ae8fff	ffb70472	c062d46f	961f4894	b21bad3d	6d83bfc6	7e44008e
62	04d24d6c	b85e2ce9	b6ae8fff	ffb70472	948d25b6	961f4894	b21bad3d	6d83bfc6
63	d39a2165	04d24d6c	b85e2ce9	b6ae8fff	fb121210	948d25b6	961f4894	b21bad3d
64	506e3058	d39a2165	04d24d6c	b85e2ce9	5ef50f24	fb121210	948d25b6	961f4894

5. 最终哈希值

H0	0xba7816bf
H1	0x8f01cfea
H2	0x414140de
H3	0x5dae2223
H4	0xb00361a3
H5	0x96177a9c
H6	0xb410ff61
H7	0xf20015ad