SHA -256

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function developed by the NSA and standardized by NIST as part of the SHA-2 family. It transforms an input of arbitrary length into a fixed 256-bit output, ensuring data integrity and authenticity.

Characteristics:-

- Produces a 256-bit (32-byte) hash value
- Processes input in 512-bit blocks
- Deterministic: same input always gives the same output
- One-way: infeasible to reverse or reconstruct the input
- Collision-resistant: hard to find two inputs with the same hash

Advantages:-

- Strong security against known cryptographic attacks
- Widely adopted and trusted standard
- Efficient and fast in both hardware and software implementations
- Suitable for hashing large data streams

Applications:-

- Digital signatures and certificates
- Password hashing and authentication
- Blockchain and cryptocurrency (e.g., Bitcoin)
- File integrity verification
- Secure communications and protocols (SSL/TLS)

Algorithm:-

Write the message in binary using ASCII coding.

1. Message Preprocessing
   a. Padding:

- Append a '1' bit to the original message
- Add k '0' bits where k is the smallest non-negative solution to:
  $(l+1+k) = 448 \bmod 512$
  (l = message length in bits)
- Append 64-bit big-endian representation of l

b. Parsing:
Divide padded message into N×512-bit blocks:
M(1),M(2),...,M(N)

2. Message Schedule Preparation

For each 512 bit block:

a. Divide into 16×32-bit words W0 to W15

b. Expand to 64 words using:
$Wt=\sigma 1(W_{t-2})+W_{t-7}+\sigma 0(W_{t-15})+W_{t-16}$
for $16 \leq t \leq 63$

Where: (lowercase sigma)
$\sigma 0(x)=ROTR7(x)\oplus ROTR18(x)\oplus SHR3(x)$
$\sigma 1(x)=ROTR17(x)\oplus ROTR19(x)\oplus SHR10(x)$

ROTR is circular right shift and SHR is shift right by padding zeroes.

3. Compression Function

Initialize working variables with initial hash values (H^0):
H0(0) = 6a09e667
H1(0) = bb67ae85
H2(0) = 3c6ef372
H3(0) = a54ff53a
H4(0) = 510e527f
H5(0) = 9b05688c
H6(0) = 1f83d9ab
H7(0) =5be0cd19
These words were obtained by taking the first thirty-two bits of the fractional parts of the square roots of the first eight prime numbers.

For each message block:

a. Initialize working variables:
a=H0(i−1),b=H1(i−1),...,h=H7(i−1)

b. Main Loop (64 iterations):
For t = 0 to 63:

1. Compute intermediate values:
   T1 = $h+\Sigma 1(e)+Ch(e,f,g)+Kt+Wt$
   T2 = $\Sigma 0(a)+Maj(a,b,c)$

   K are 64 constants of 32 bits each representing the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four prime numbers.

2. Update working variables:
   h=g
   g=f
   f=e
   e=d+T1
   d=c
   c=b
   b=a
   a=T1+T2

Where:
$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$, (here ^ is bitwise AND)
$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
$\Sigma 0(x) = ROTR2(x) \oplus ROTR13(x) \oplus ROTR22(x)$
$\Sigma 1(x) = ROTR6(x) \oplus ROTR11(x) \oplus ROTR25(x)$

c. Final Update:
H0(i)=a+H0(i−1)
H1(i)=b+H1(i−1)
...
H7(i)=h+H7(i−1)

4. Output Construction
After processing all blocks:
Digest=H0(N)||H1(N)||...||H7(N)

Verilog Implementation:-

10 stage pipelining is used.

Stage 1: On reset, message block would be defined and initial hash values are stored as intermediate state. Words W0 to W15 are calculated.

Stage 2: Round R0 to R7 calculated and word W16 to W22

Stage 3: Round R7 to R13 calculated and word W23 to W29

Stage 4: Round R14 to R20 calculated and word W30 to W36

Stage 5: Round R21 to R27 calculated and word W37 to W43

Stage 6: Round R28 to R34 calculated and word W44 to W50

Stage 7: Round R35 to R41 calculated and word W51 to W57

Stage 8: Round R42 to R48 calculated and word W58 to W63

Stage 9: Round R49 to R55 calculated

Stage 10: Round R56 to R63 calculated and final values of hash stored

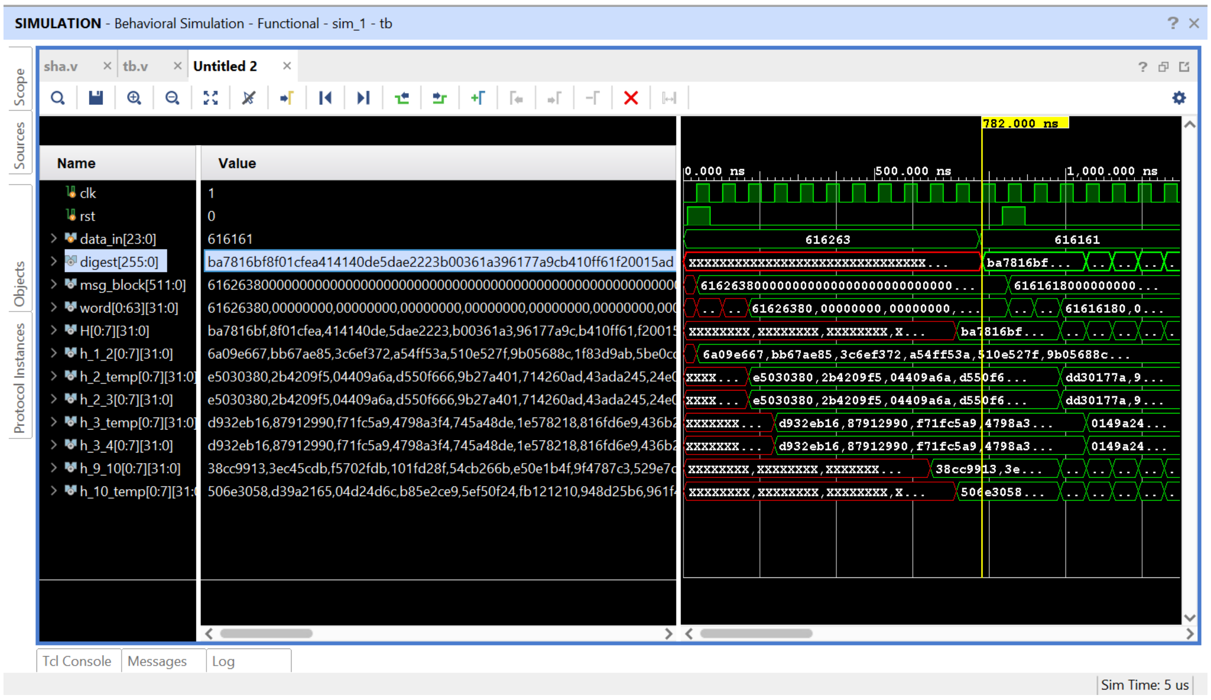
The message block is initialised only when reset is set. We get final hash value 10 clock cycles after reset is set.

Temporary registers for hash values are used between stages so as to not overwrite.

Clock frequency of 14.705MHz is used with time period of 68ns.

Simulation and Results:-

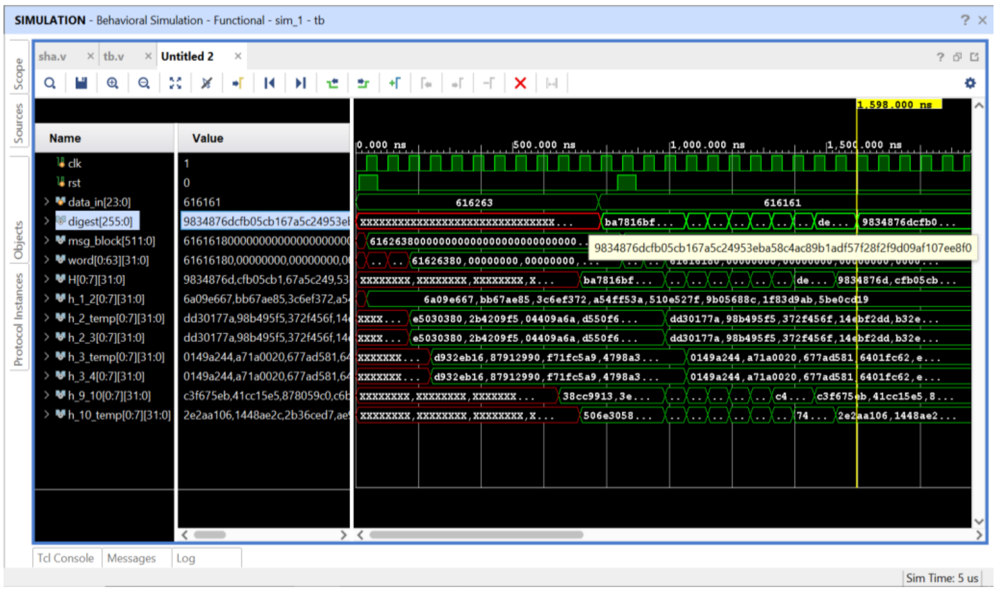
Input was "abc" which in hex is "616263". Its output hash value comes to be ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad


SIMULATION - Behavioral Simulation - Functional - sim_1 - tb

sha.v | tb.v | Untitled 2

Name	Value
clk	1
rst	0
data_in[23:0]	616161
digest[255:0]	ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
msg_block[511:0]	6162638000000000000000000000000000000000000000000000000000000000
word[0:63][31:0]	61626380,00000000,00000000,00000000,00000000,00000000,00000000,000
H[0:7][31:0]	ba7816bf,8f01cfea,414140de,5dae2223,b00361a3,96177a9c,b410ff61,f20015
h_1_2[0:7][31:0]	6a09e667,bb67ae85,3c6ef372,a54ff53a,510e527f,9b05688c,1f83d9ab,5be0cd
h_2_temp[0:7][31:0]	e5030380,2b4209f5,04409a6a,d550f666,9b27a401,714260ad,43ada245,24e0
h_2_3[0:7][31:0]	e5030380,2b4209f5,04409a6a,d550f666,9b27a401,714260ad,43ada245,24e0
h_3_temp[0:7][31:0]	d932eb16,87912990,f71fc5a9,4798a3f4,745a48de,1e578218,816fd6e9,436b2
h_3_4[0:7][31:0]	d932eb16,87912990,f71fc5a9,4798a3f4,745a48de,1e578218,816fd6e9,436b2
h_9_10[0:7][31:0]	38cc9913,3ec45cdb,f5702fdb,101fd28f,54cb266b,e50e1b4f,9f4787c3,529e7c
h_10_temp[0:7][31:	506e3058,d39a2165,04d24d6c,b85e2ce9,5ef50f24,fb121210,948d25b6,961f4

When another input is given "aaa" whose hex is "616161", its output comes to be 9834876dcfb05cb167a5c24953eba58c4ac89b1adf57f28f2f9d09af107ee8f0



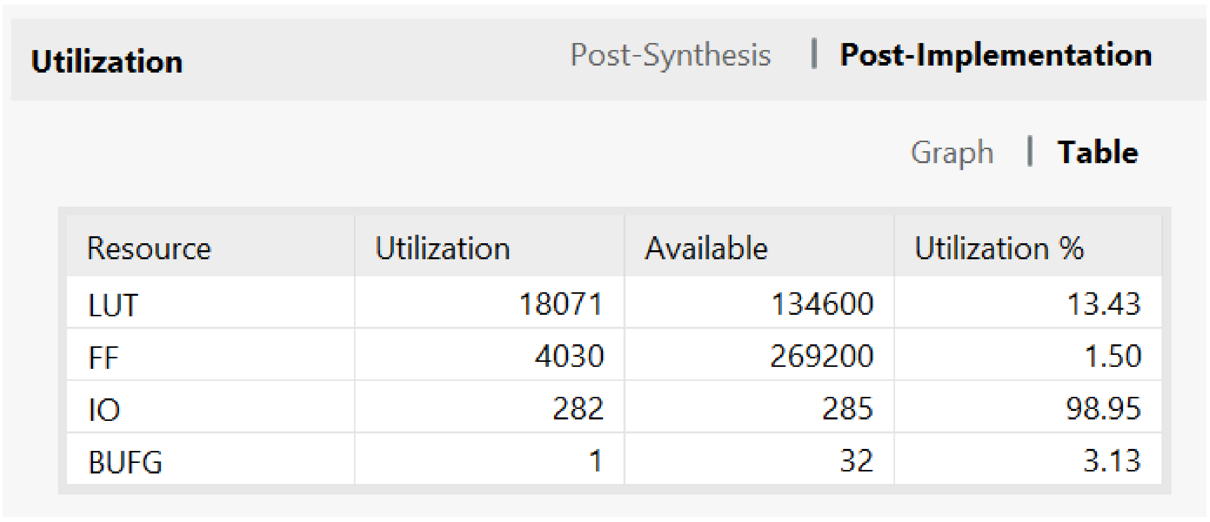
SIMULATION - Behavioral Simulation - Functional - sim_1 - tb

Name	Value
clk	1
rst	0
data_in[23:0]	616161
digest[255:0]	9834876dcfb05cb167a5c24953eb
msg_block[511:0]	61616180000000000000000000
word[0:63][31:0]	61616180,00000000,00000000,0
H[0:7][31:0]	9834876d,cfb05cb1,67a5c249,53
h_1_2[0:7][31:0]	6a09e667,bb67ae85,3c6ef372,a5
h_2_temp[0:7][31:0]	dd30177a,98b495f5,372f456f,14
h_2_3[0:7][31:0]	dd30177a,98b495f5,372f456f,14
h_3_temp[0:7][31:0]	0149a244,a71a0020,677ad581,64
h_3_4[0:7][31:0]	0149a244,a71a0020,677ad581,64
h_9_10[0:7][31:0]	c3f675eb,41cc15e5,878059c0,c6
h_10_temp[0:7][31:0]	2e2aa106,1448ae2c,2b36ced7,ae

9834876dcfb05cb167a5c24953eba58c4ac89b1adf57f28f2f9d09af107ee8f0

Power used is 924.948W.

Resources used are



Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	18071	134600	13.43
FF	4030	269200	1.50
IO	282	285	98.95
BUFG	1	32	3.13