ECE 564 – SHA 256 Report

/10

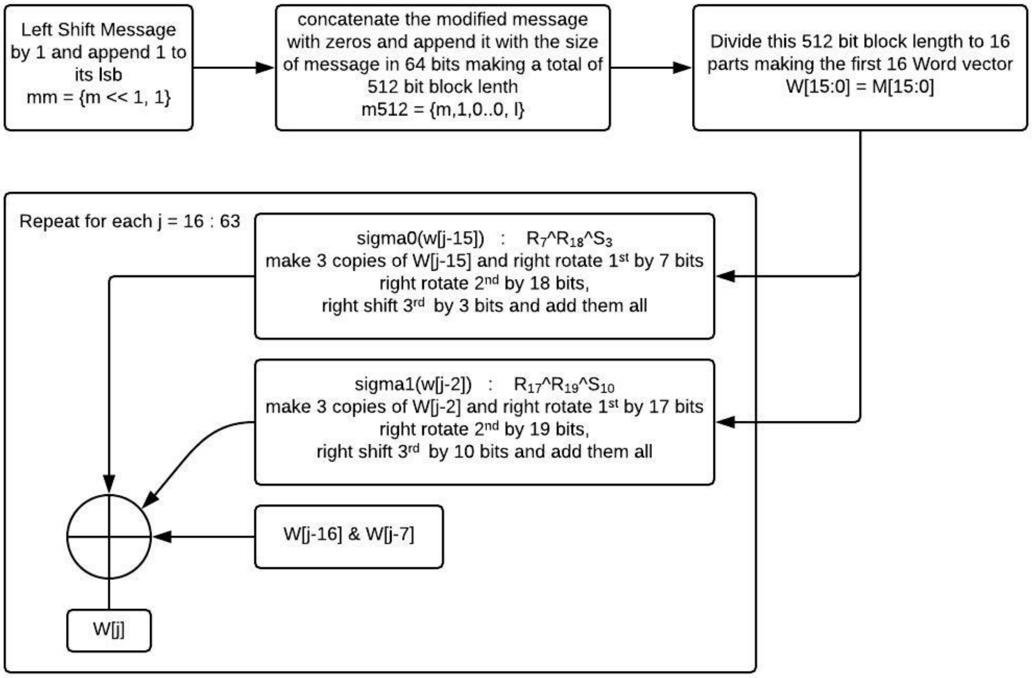
Name: Vishal Ganesh Shitole
Unity ID: vgshitol
Student ID: 200258381

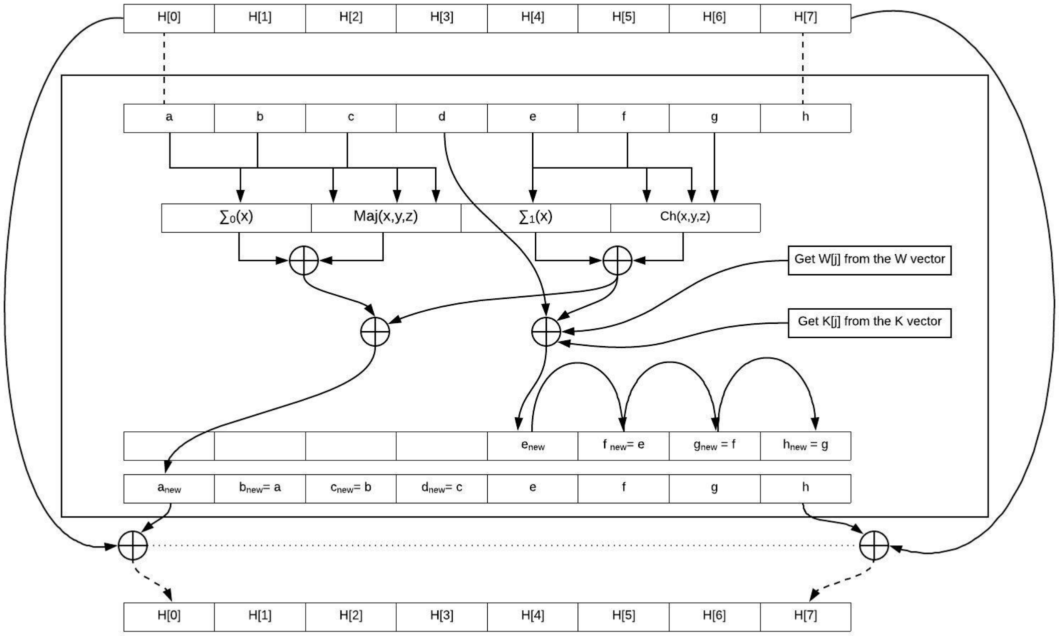
Summary Risk Plan:
Understanding the flow of design. Design before coding.
Debugging Complexity
Repeating logic for 64 word vectors
Serial read to parallel processing synchronization

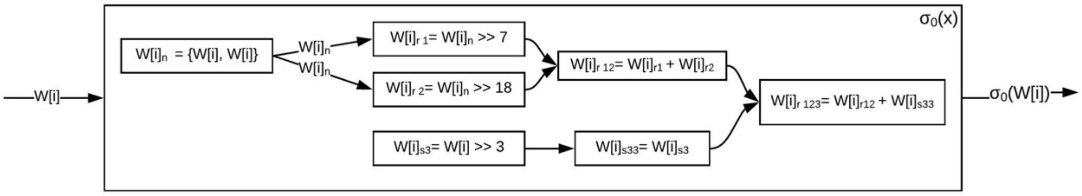
Schedule:
3 Days to form Message vector
1 week to form W vector
1 week to form final hash
1 week debugging and final report submission

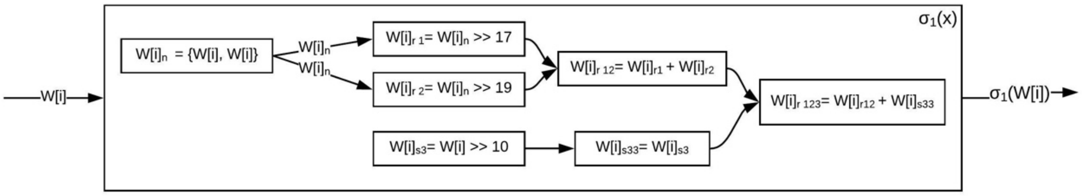
Brief Description of Mode of operation, including selected algorithms
Read data serially from SRAM and store it in register.
Form the 64 block vector
Use this data to calculate the hash using the algorithm.
Update the new hash.

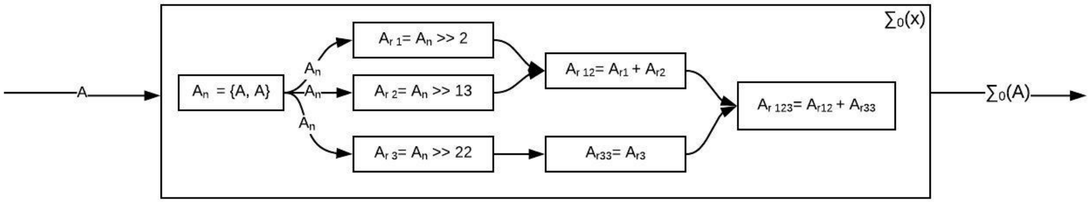
High level Design

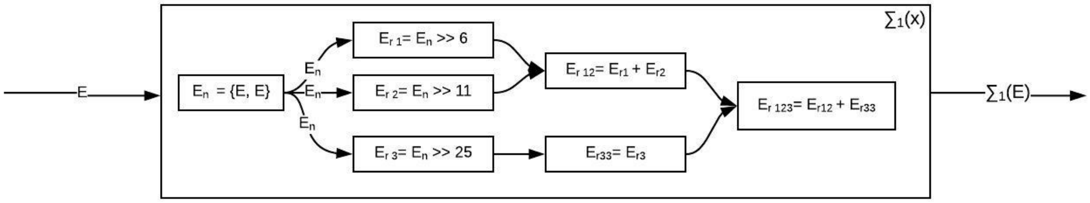

Left Shift Message by 1 and append 1 to its lsb
mm = {m << 1, 1}
concatenate the modified message with zeros and append it with the size of message in 64 bits making a total of 512 bit block lenth
m512 = {m,1,0..0, l}
Divide this 512 bit block length to 16 parts making the first 16 Word vector
W[15:0] = M[15:0]
Repeat for each j = 16 : 63
sigma0(w[j-15]) : R_7^R_{18}^S_3
make 3 copies of W[j-15] and right rotate 1st by 7 bits
right rotate 2nd by 18 bits,
right shift 3rd by 3 bits and add them all
sigma1(w[j-2]) : R_{17}^R_{19}^S_{10}
make 3 copies of W[j-2] and right rotate 1st by 17 bits
right rotate 2nd by 19 bits,
right shift 3rd by 10 bits and add them all
W[j-16] & W[j-7]
W[j]

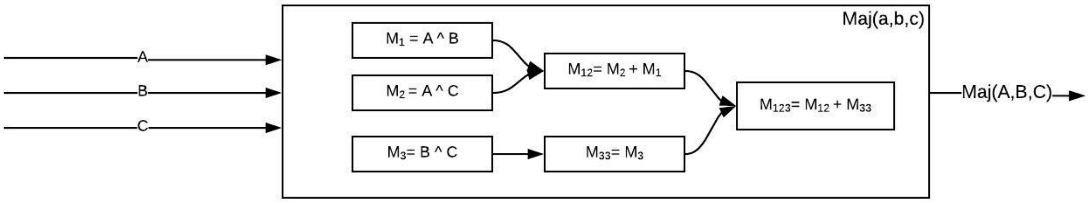

H[0]
H[1]
H[2]
H[3]
H[4]
H[5]
H[6]
H[7]
a
b
c
d
e
f
g
h
$\sum_0(x)$
Maj(x,y,z)
$\sum_1(x)$
Ch(x,y,z)
Get W[j] from the W vector
Get K[j] from the K vector
e_{new}
f_{new}= e
g_{new} = f
h_{new} = g
a_{new}
b_{new}= a
c_{new}= b
d_{new}= c
e
f
g
h
H[0]
H[1]
H[2]
H[3]
H[4]
H[5]
H[6]
H[7]

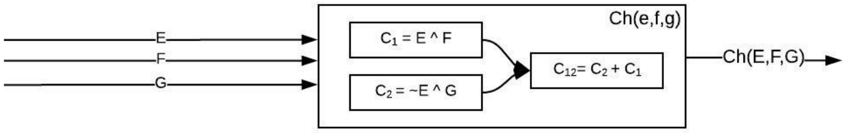

$\sigma_0(x)$
W[i]
$W[i]_n$ = {W[i], W[i]}
$W[i]_n$
$W[i]_n$
$W[i]_{r1}$= $W[i]_n$ >> 7
$W[i]_{r2}$= $W[i]_n$ >> 18
$W[i]_{s3}$= W[i] >> 3
$W[i]_{r12}$= $W[i]_{r1}$ + $W[i]_{r2}$
$W[i]_{s33}$= $W[i]_{s3}$
$W[i]_{r123}$= $W[i]_{r12}$ + $W[i]_{s33}$
σ_0(W[i])


$\sigma_1(x)$
W[i]
$W[i]_n$ = {W[i], W[i]}
$W[i]_n$
$W[i]_n$
$W[i]_{r1}$= $W[i]_n$ >> 17
$W[i]_{r2}$= $W[i]_n$ >> 19
$W[i]_{s3}$= W[i] >> 10
$W[i]_{r12}$= $W[i]_{r1}$ + $W[i]_{r2}$
$W[i]_{s33}$= $W[i]_{s3}$
$W[i]_{r123}$= $W[i]_{r12}$ + $W[i]_{s33}$
σ_1(W[i])


$\sum_0(x)$
A
A_n = {A, A}
A_n
A_n
A_n
A_{r1}= A_n >> 2
A_{r2}= A_n >> 13
A_{r3}= A_n >> 22
A_{r12}= A_{r1} + A_{r2}
A_{r33}= A_{r3}
A_{r123}= A_{r12} + A_{r33}
$\sum_0$(A)


$\sum_1(x)$
E
E_n = {E, E}
E_n
E_n
E_n
E_{r1}= E_n >> 6
E_{r2}= E_n >> 11
E_{r3}= E_n >> 25
E_{r12}= E_{r1} + E_{r2}
E_{r33}= E_{r3}
E_{r123}= E_{r12} + E_{r33}
$\sum_1$(E)





Name: Vishal Ganesh Shitole
Unityid: vgshitol
StudentID: 200258381

Delay (ns to run provided provided example). Clock period: 24 ns # cycles”: 84	Logic Area: (um^2) **36542.01** Memory: N/A	1/(delay.area) (ns^{-1}.um^{-2}) 1.14024 * 10^{-6}

Delay (TA provided example. TA to complete)	1/(delay.area) (TA)

Abstract

This Hardware Module Encrypts the message in SHA256 which is the standard encryption technique. The input message can be 1-55 bytes message string whereas the output message string is 256 bits encrypted unique hash for that particular bit. The module implements a standard logic in 4 main parts. Getting the message, K constants and initial Hash Vector, forming the Word Vector, Updating the Hashes and the Storing the encrypted hash in the SRAM. Each of the above four sections are executed by dedicated submodules integrated together to form the encrypted hash. We see,

1. Introduction

The SHA (Secure Hash Algorithm) is one of a number of **cryptographic hash functions**. A cryptographic hash is like a signature for a text or a data file. SHA-256 algorithm generates an almost-unique, fixed size 256-bit (32-byte) hash. Hash is a one way function – it cannot be decrypted back. This makes it suitable for password validation, challenge hash authentication, anti-tamper, digital signatures. To design this hardware module we split it into four main parts i.e. Getting the message, K constants and initial Hash Vector, forming the Word Vector, Updating the Hashes and the Storing the encrypted hash in the SRAM.

We see that the module needs cell area around 32,000 and runs without violation on a clock of 24 ns. Total time required for the generation of the encrypted hash key is 120 cycles.

The report further illustrates the algorithm used and the design approach, implementation and verification methodologies.

2. Micro-Architecture

- Hardware Algorithmic Approach:

The algorithm is designed in the form of modules.

i. "msgEn": This module sets the enable signal high for processing the following sections. It takes a signal input stable and sets the output high until the next reset occurs.

ii. "counter": This module is a standard up counter and counts the address/index of the Message Vectors. The module has a parameter Number of Blocks which is used to set the maximum counter limit. The module takes input maximum message length, start flag which comes from the "msgEn" module and outputs the address/index and the completeFlag indicating that the counter is complete. After complete the counter latches the last value till the enable is high.

iii. "msg_vector": This module constructs the 512 bit Message Vector. The module takes input enable flag which comes from the output of "msgEn" module, address read complete flag and address/index which comes from the output of the "counter" and msg_data which is the 8 bit data to form the msg_vector. The module outputs the msg_write flag indicating that it is a read instruction or a write, the 512 bit message_vector and the completeFlag indicating that the msg_vector is complete. After complete the msg_vector module latches the msg_vector till the enable is high.

iv. "counter_h": This module is a standard up counter and counts the address/index of the Hash Vector. The module has a parameter Number of Blocks which is used to set the counter limit. The module takes input start flag which comes from the "msgEn" module and outputs the address/index and the completeFlag indicating that the counter is complete. After complete the counter latches the last value till the enable is high.

v. "hash": This module constructs the initial 256 bit Hash Vector. The module takes input enable flag which comes from the output of "msgEn" module, address read complete flag and address/index which comes from the output of the "counter_h" and hash_data which is the 8 bit data to form the hash. The module outputs the hash_write flag indicating that it is a read instruction or a write, the 256 bit hash_vector and the completeFlag indicating that the hash_vector is complete. After "complete" the hash module latches the hash_vector till the enable is high.

vi. "counter_wk": This module is a standard up counter and counts the address/index of the Word and K Values. The module has a parameter Number of Blocks whose default is 64 which is used to set the counter limit. The module takes input start flag which comes from the "msgEn" module and outputs the address/index and the completeFlag indicating that the counter is complete. After complete the counter latches the last value till the enable is high.

vii. "wEn": This module sets the enable signal high for processing the following sections. It takes 2 signal inputs, one to indicate message vector formation and other to indicate the hash vector formation and sets the output high if both the input signals are high.

viii. "k_vector": This module constructs the 32 bit K Value. The module takes input enable flag which comes from the output of "wEn" module, address read complete flag and address/index value which comes from the output of the "counter_wk" and k_data which is the 32 bit data. The module outputs the k_write flag indicating that it is a read instruction or a write, the 32 bit current K Value and the completeFlag indicating that the last K value is read. After "complete" the k_vector module latches the last k value till the enable is high.

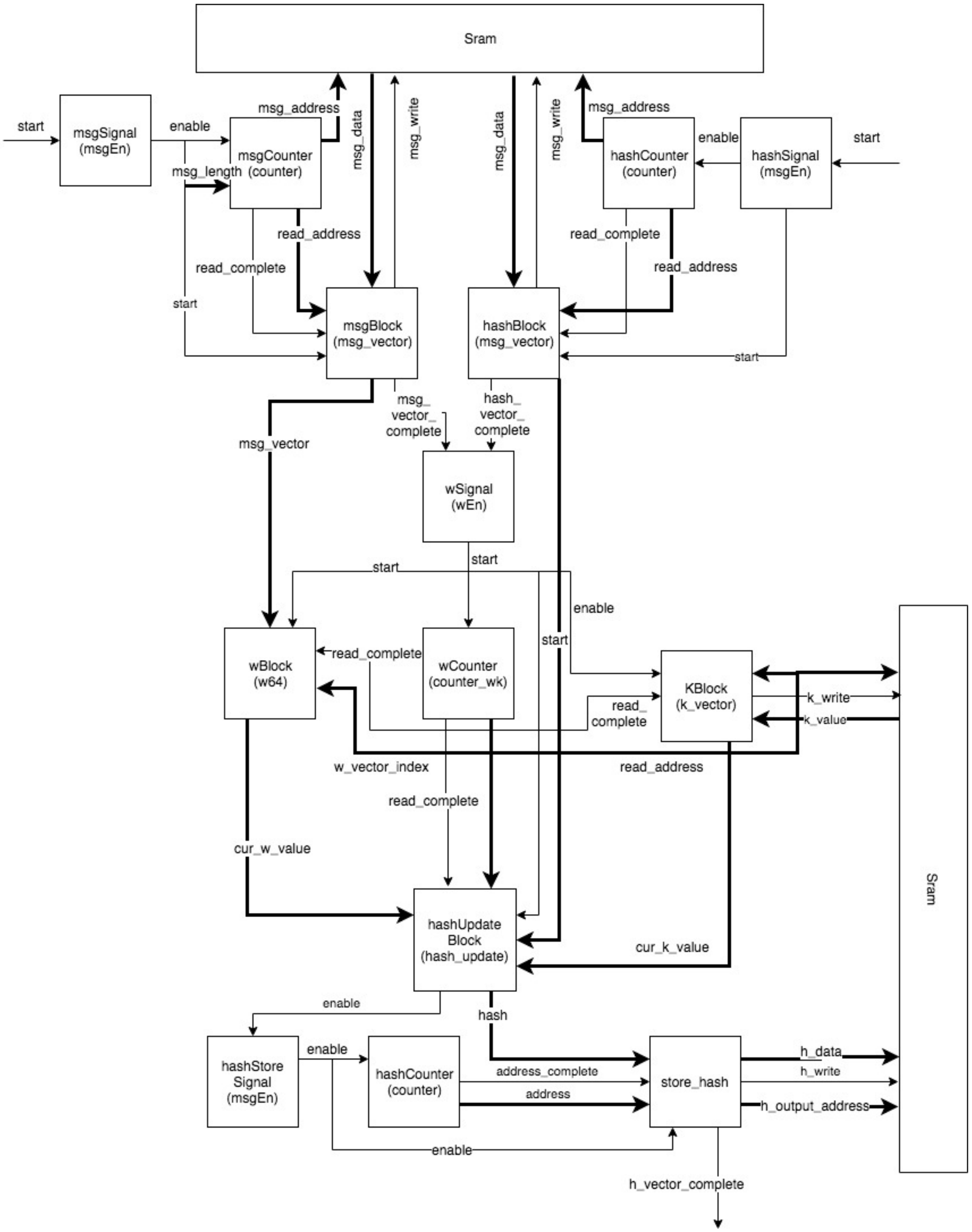
ix. "w64": This module constructs the 32 bit Word and the Word vector which is stored internally. The module takes input enable flag which comes from the output of "wEn" module, address read complete flag and address/index value which comes from the output of the "counter_wk" and message_vector which is the 512 bit message formed by the msg_vector module. The module outputs the 32 bit current W Value and the complete Flag indicating that the last W value is read. After "complete" the w64 module latches the last w value till the enable is high.

x. "hash_update": This module updates the 256 bit hash until the 64th word. The module takes input enable flag which comes from the output of "wEn" module, address read complete flag and address/index value which comes from the output of the "counter_wk" and cur_w_value and cur_k_value (both 32 bits) formed by the k_vector and w64 module. The module outputs the 256 bit current hash update Value and the complete Flag indicating that the last hash update is complete. After "complete" the hash_update module latches the 256 bit hash value till the enable is high.

xi. "store_hash": This module stores the resultant 256 bit updated or encrypted Hash Vector. The module takes input enable flag which comes from the output of "msgEn" module, address read complete flag and address/index which comes from the output of the "counter_h" and the 256 bit hash_vector. The module outputs the hash_data which is the 8 bit data hash, hash_write flag indicating that it is a write, and the complete Flag indicating that the hash storage is complete.

3. Interface Specification

- Interface Diagram/ Module Diagram



- Module IO ports Table

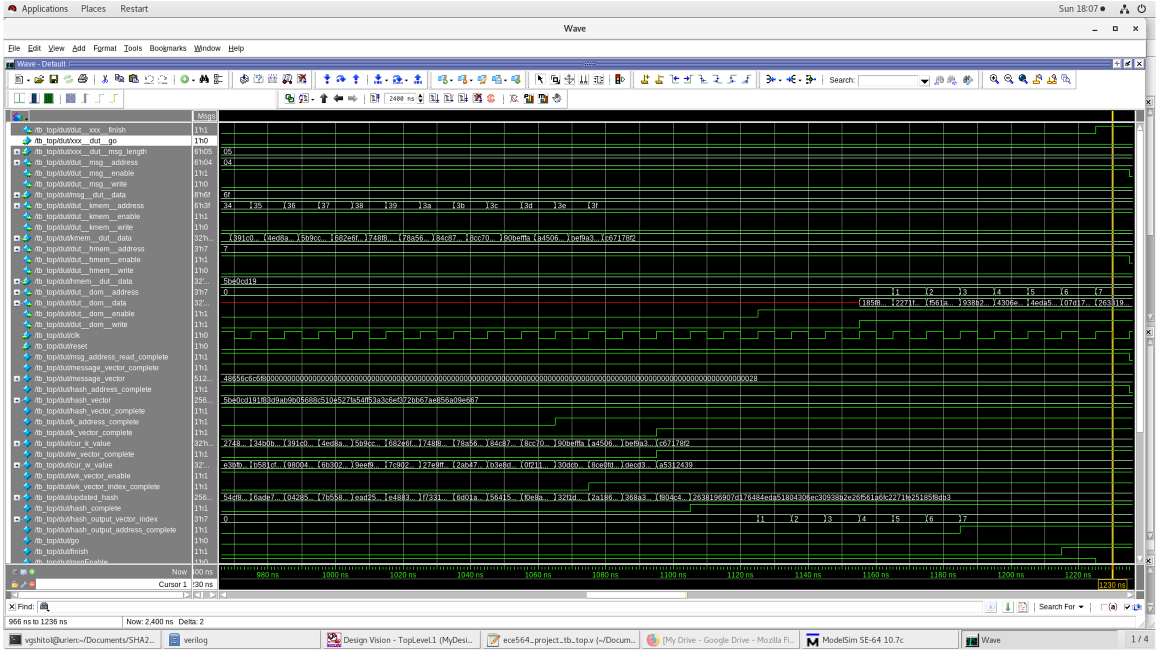
Sr No.	Inputs and Outputs	Width	Function
Module: msgEn			
1.	Start (input)	1	Used as starting point for the functions
2.	Enable (output)	1	Used to enable modules
Module: counter			
1.	Start (input)	1	Used as starting point for the function
2.	Msg_length (input)	6	Used to compare the counter length
3.	Read_address (output)	6	Used to specify the address index of the vector
4.	Read_complete (output)	1	Used to detect if the address index has reached its limit and complete
Module: msg_vector			
1.	enable (input)	1	Used as starting point for the function
2.	Address_read_complete (input)	1	Used as end point for the function
3.	Msg_address (input)	6	Used to specify the address index of the vector
4.	Msg_data (input)	8	Gets the Message data bytes
5.	Msg_write (output)	1	Set 0 for Reading the message data
6.	Message_vector (output)	512	Get the 512 Bit message vector.
7.	Msg_vector_complete (output)	1	Detects the Completion of Message vector formation
Module: counter_h			
1.	Start (input)	1	Used as starting point for the function
2.	Read_address (output)	6	Used to specify the address index of the vector
3.	Read_complete (output)	1	Used to detect if the address index has reached its limit and complete
Module: wEn			
1.	Start1 (input)	1	Used as starting point for the functions
2.	Start2 (input)	1	Used as starting point for the functions
3.	Enable (output)	1	Used to enable modules
Module: hash			
1.	enable (input)	1	Used as starting point for the function
2.	Address_read_complete (input)	1	Used as end point for the function
3.	hash_address (input)	3	Used to specify the address index of the vector
4.	hash_data (input)	8	Gets the Hash data bytes
5.	hash_write (output)	1	Set 0 for Reading the message data
6.	hash_vector (output)	256	Get the 256 Bit hash vector.
7.	hash_vector_complete (output)	1	Detects the Completion of Hash vector formation
Module: k_vector			

1.	enable (input)	1	Used as starting point for the function
2.	Address_read_complete (input)	1	Used as end point for the function
3.	k_address (input)	6	Used to specify the address index of the vector
4.	k_data (input)	32	Gets the K word of 32 bits
5.	k_write (output)	1	Set 0 for Reading the message data
6.	Cur_k_value (output)	32	Get the 32 Bit current K value.
7.	hash_vector_complete (output)	1	Detects K vector has been read fully
Module: w64			
1.	enable (input)	1	Used as starting point for the function
2.	W_index_complete (input)	1	Used as end point for the function
3.	Message_vector (input)	512	Message Vector that is used to form the Word Vector
4.	w_vector_index (input)	6	Used to specify the address index of the vector
5.	Cur_w_value (output)	32	Get the 32 Bit current Word.
6.	w_vector_complete (output)	1	Detects W vector formation has been completed
Module: hash_update			
1.	enable (input)	1	Used as starting point for the function
2.	WK_index_complete (input)	1	Used as end point of the indexing
3.	Prev_hash (input)	256	Initial Hash that is used to update and encrypt SHA256
4.	wk_vector_index (input)	6	Used to specify the address index of the vector
5.	Cur_w_value (output)	32	Get the 32 Bit current Word.
6.	Cur_k_value (output)	32	Get the 32 Bit current K Value.
7.	hash_complete (output)	1	Detects hash update has been completed
Module: store_hash			
1.	enable (input)	1	Used as starting point for the function
2.	Address_read_complete (input)	1	Used as end point for the function
3.	h_address (input)	3	Used to specify the address index of the vector
4.	hash_vector (input)	256	Hash vector needed to be stored in sram
5.	h_write (output)	1	Set 1 for writing the hash data
6.	h_data (output)	32	Extract the 32 Bit hash data part by part to store it in sram.
7.	H_output_address (output)	3	Needed to Extract the 32 Bit hash data part by part to store it in sram.
8.	hash_vector_complete (output)	1	Detects the Completion of Hash vector formation

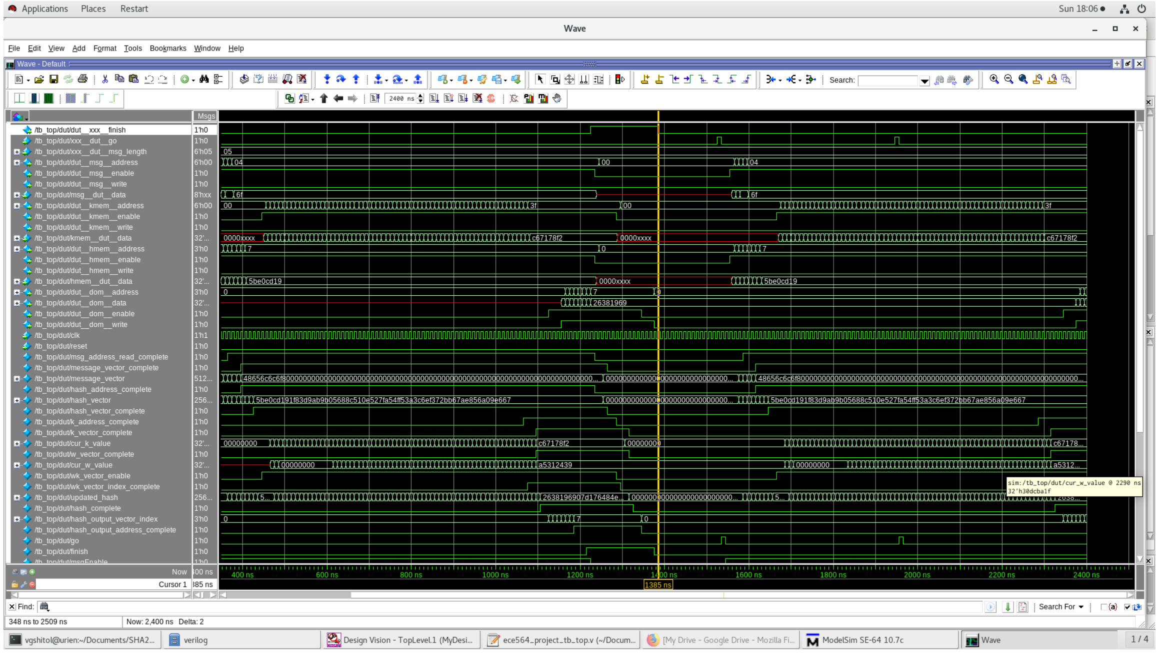
4. Technical Implementation – High Level Description and Hierarchy

As stated in the previous sections we follow the modular approach for designing our required functionality. We execute every stage with it's enable signal, a counter for its index referencing and the main processing block/module. The enable module only sets the enable flag which enables the counter and the process block. The counter up-counts till the maximum limit and stops at the last index latching the output. The process block executes step by step until the counter runs and completes when the counter is complete. Message and Hash Blocks run in parallel. When both of them are complete the W block, K block and the Hash Update Block run in parallel with the counter WK-Counter. Once the Hash Update is complete, we enable the store functionality which runs its counter and store the updated hash step by step.

5. Verification


Applications Places Restart
Sun 18:07
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_top/dut/dut__xxx__finish 1'h1
/tb_top/dut/xxx__dut__go 1'h0
/tb_top/dut/xxx__dut__msg_length 6'h05 05
/tb_top/dut/dut__msg__address 6'h04 04
/tb_top/dut/dut__msg__enable 1'h1
/tb_top/dut/dut__msg__write 1'h0
/tb_top/dut/msg__dut__data 8'h6f 6f
/tb_top/dut/dut__kmem__address 6'h3f 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f
/tb_top/dut/dut__kmem__enable 1'h1
/tb_top/dut/dut__kmem__write 1'h0
/tb_top/dut/kmem__dut__data 32'h... 391c0... 4ed8a... 5b9cc... 682e6f... 748f8... 78a56... 84c87... 8cc70... 90befffa a4506... bef9a3... c67178f2
/tb_top/dut/dut__hmem__address 3'h7 7
/tb_top/dut/dut__hmem__enable 1'h1
/tb_top/dut/dut__hmem__write 1'h0
/tb_top/dut/hmem__dut__data 32'... 5be0cd19
/tb_top/dut/dut__dom__address 3'h7 0 1 2 3 4 5 6 7
/tb_top/dut/dut__dom__data 32'... 185f8... 2271f... f561a... 938b2... 4306e... 4eda5... 07d17... 263819...
/tb_top/dut/dut__dom__enable 1'h1
/tb_top/dut/dut__dom__write 1'h1
/tb_top/dut/clk 1'h0
/tb_top/dut/reset 1'h0
/tb_top/dut/msg_address_read_complete 1'h1
/tb_top/dut/message_vector_complete 1'h1
/tb_top/dut/message_vector 512... 48656c6c6f800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000028
/tb_top/dut/hash_address_complete 1'h1
/tb_top/dut/hash_vector 256... 5be0cd191f83d9ab9b05688c510e527fa54ff53a3c6ef372bb67ae856a09e667
/tb_top/dut/hash_vector_complete 1'h1
/tb_top/dut/k_address_complete 1'h1
/tb_top/dut/k_vector_complete 1'h1
/tb_top/dut/cur_k_value 32'h... 2748... 34b0b... 391c0... 4ed8a... 5b9cc... 682e6f... 748f8... 78a56... 84c87... 8cc70... 90befffa a4506... bef9a3... c67178f2
/tb_top/dut/w_vector_complete 1'h1
/tb_top/dut/cur_w_value 32'... e3bfb... b581cf... 98004... 6b302... 9eef9... 7c902... 27e9ff... 2ab47... b3e8d... 0f211... 30dcb... 8ce0fd... decd3... a5312439
/tb_top/dut/wk_vector_enable 1'h1
/tb_top/dut/wk_vector_index_complete 1'h1
/tb_top/dut/updated_hash 256... 54cf8... 6ade7... 04285... 7b558... ead25... e4883... f7331... 6d01a... 56415... f0e8a... 32f1d... 2a186... 368a3... f804c4... 2638196907d176484eda51804306ec30938b2e26f561a6fc2271fe25185f8db3
/tb_top/dut/hash_complete 1'h1
/tb_top/dut/hash_output_vector_index 3'h7 0 1 2 3 4 5 6 7
/tb_top/dut/hash_output_address_complete 1'h1
/tb_top/dut/go 1'h0
/tb_top/dut/finish 1'h1
/tb_top/dut/msgEnable 1'h0
Now 2400 ns
Cursor 1 1230 ns
980 ns 1000 ns 1020 ns 1040 ns 1060 ns 1080 ns 1100 ns 1120 ns 1140 ns 1160 ns 1180 ns 1200 ns 1220 ns
1230 ns
Find:
Search For
966 ns to 1236 ns
Now: 2,400 ns Delta: 2
vgshitol@urien:~/Documents/SHA2...
verilog
Design Vision - TopLevel.1 (MyDesi...
ece564_project_tb_top.v (~/Docum...
[My Drive - Google Drive - Mozilla Fi...
ModelSim SE-64 10.7c
Wave
1 / 4

From the above screenshot we can verify that our hash vector is formed and stored correctly.


Applications Places Restart
Sun 18:06
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/tb_top/dut/dut__xxx__finish 1'h0
/tb_top/dut/xxx__dut__go 1'h0
/tb_top/dut/xxx__dut__msg_length 6'h05
/tb_top/dut/dut__msg__address 6'h00
/tb_top/dut/dut__msg__enable 1'h0
/tb_top/dut/dut__msg__write 1'h0
/tb_top/dut/msg__dut__data 8'hxx
/tb_top/dut/dut__kmem__address 6'h00
/tb_top/dut/dut__kmem__enable 1'h0
/tb_top/dut/dut__kmem__write 1'h0
/tb_top/dut/kmem__dut__data 32'...
/tb_top/dut/dut__hmem__address 3'h0
/tb_top/dut/dut__hmem__enable 1'h0
/tb_top/dut/dut__hmem__write 1'h0
/tb_top/dut/hmem__dut__data 32'...
/tb_top/dut/dut__dom__address 3'h0
/tb_top/dut/dut__dom__data 32'...
/tb_top/dut/dut__dom__enable 1'h0
/tb_top/dut/dut__dom__write 1'h0
/tb_top/dut/clk 1'h1
/tb_top/dut/reset 1'h0
/tb_top/dut/msg_address_read_complete 1'h0
/tb_top/dut/message_vector_complete 1'h0
/tb_top/dut/message_vector 512...
/tb_top/dut/hash_address_complete 1'h0
/tb_top/dut/hash_vector 256...
/tb_top/dut/hash_vector_complete 1'h0
/tb_top/dut/k_address_complete 1'h0
/tb_top/dut/k_vector_complete 1'h0
/tb_top/dut/cur_k_value 32'...
/tb_top/dut/w_vector_complete 1'h0
/tb_top/dut/cur_w_value 32'...
/tb_top/dut/wk_vector_enable 1'h0
/tb_top/dut/wk_vector_index_complete 1'h0
/tb_top/dut/updated_hash 256...
/tb_top/dut/hash_complete 1'h0
/tb_top/dut/hash_output_vector_index 3'h0
/tb_top/dut/hash_output_address_complete 1'h0
/tb_top/dut/go 1'h0
/tb_top/dut/finish 1'h0
/tb_top/dut/msgEnable 1'h0
Now 400 ns
Cursor 1 385 ns
400 ns
600 ns
800 ns
1000 ns
1200 ns
1400 ns
1600 ns
1800 ns
2000 ns
2200 ns
2400 ns
1385 ns
Find:
Search For
348 ns to 2509 ns
Now: 2,400 ns Delta: 2
vgshitol@urien:~/Documents/SHA2...
verilog
Design Vision - TopLevel.1 (MyDesi...
ece564_project_tb_top.v (~/Docum...
[My Drive - Google Drive - Mozilla Fi...
ModelSim SE-64 10.7c
Wave
1 / 4

From the above screenshot it is evident that the message signal does not respond to intermediate go signals. Once the hash formation of the previous go is complete, the module accepts a new request and new hash is created. We see a delay of some finite clock cycles is required between two consecutive go signals for the system to detect a new request instead of ignoring it.

6. Results Achieved

i. We see that for the message "Hello", we complete the hash in **84** cycles.

ii. We see that the clock for no violation after synthesis is **24ns**. This is mainly due to the increased logic in the W process Block and the Hash Update Process Block.

iii. We see that the cell are comes out to be **36542.01 um^2.**

7. Conclusion

We achieve our desired functionality of hash. We are able to get 256 bit updated hash at the expense of **36542.01** um^2