SHA-256 Hash function implementation

Name: Emil Prisquilas Peter
Unityid:epeter
StudentID:200269166

Delay (ns to run provided example). Clock period:20 ns # cycles:441(for the test bench provided)	Logic Area: (um^2) 75283.0549 Memory: N/A	1/(delay.area) ($ns^{-1}.um^{-2}$) 664.16 x 10^{-9} $ns^{-1}.um^{-2}$
Delay (TA provided example. TA to complete)		1/(delay.area) (TA)

Abstract

The main objective of the project is to implement SHA-256 algorithm to generate hash code for a given message input. The hardware is designed with the constraint of the message length being restricted to 55 characters. The hardware reads the input message, K and H from message, K and H memory respectively. K and H arrays are used for the computation of the hash code for a given message input. Various standard operations are used for computation like $\sigma_1(x)$, $\sigma_2(x)$, $\xi_1(x)$, $\xi_0(x)$, Maj (x.y,z) and Ch(x,y,z), which were a combination of XOR and shift/rotate operations. The final hash generated after computation is then written to an output memory. A finish signal was set high after the completion of write operation to output memory while a go signal going high initiates the computation of hash code. The hardware was designed such that it rejected any interrupting go signals while the computation was being performed, thus preventing errors in the final hash.

SHA-256 Hash function implementation

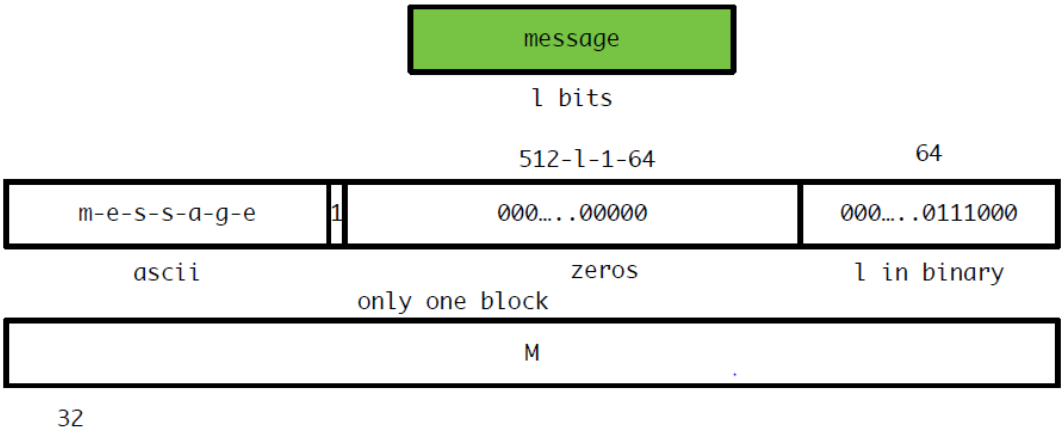
Emil Prisquilas Peter

1. Introduction

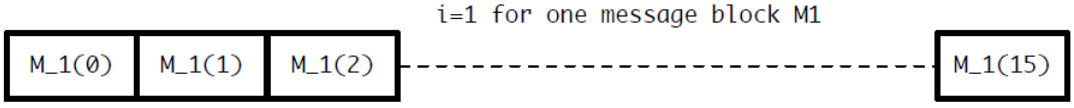
A hardware capable of generating hash code, using SHA-256 algorithm, for a given input of message is designed. The hardware was designed with the constraint of message length restricted to 55 characters. The hardware reads in the message from the predefined message memory and creates a 512 –bit M block, which in turn was used to create a M_1 array of 16 elements. W vector of 64 elements was created from the M_1 array using standard operations. K and H vector required for hash generation are read from the K and H memory respectively. Standard defined operations are performed on the elements of H vector using the elements of H, K and W. Finally the result from those operations are combined with the initial values of H vector to generate Hash code which is in turn written to an output memory.
Design was successfully simulated for messages of different lengths ranging from 1-55 characters and the results cross-checked. The hardware was successfully synthesized with an area of 75283.0549 um^2, completing the operations in 441cycles, for the given test bench with the clock period of 20 ns. The performance number obtained is 663.99 x 10^6 um^2 ns.

2. Micro-Architecture

1) The message is read from the message memory and a 512-bit M block is constructed from the message read.



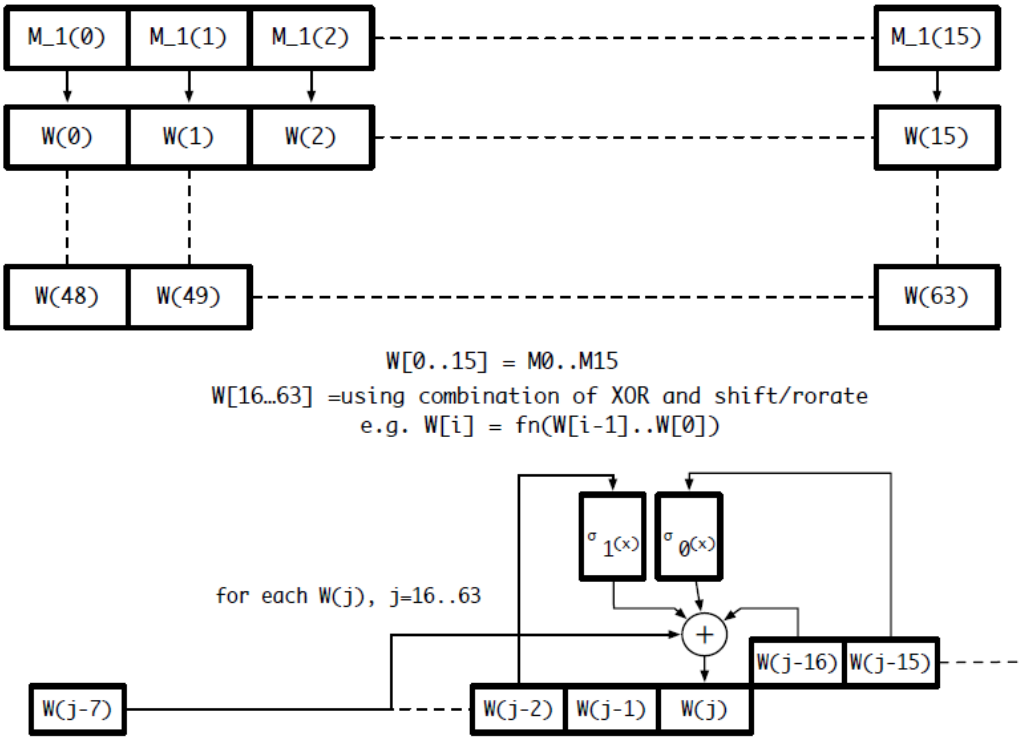
2) An array,M_1 of 16 elements of 32-bits each is created from the M block.



3) The first 16 elements of 64 elements long W array is filled with the elements of M_1 array. Rest of the elements are processed using a combination of XOR and

shift/rotate applied on the first 16 elements. Each element from 16 to 64 are treated as a function of the first 16 elements i.e.

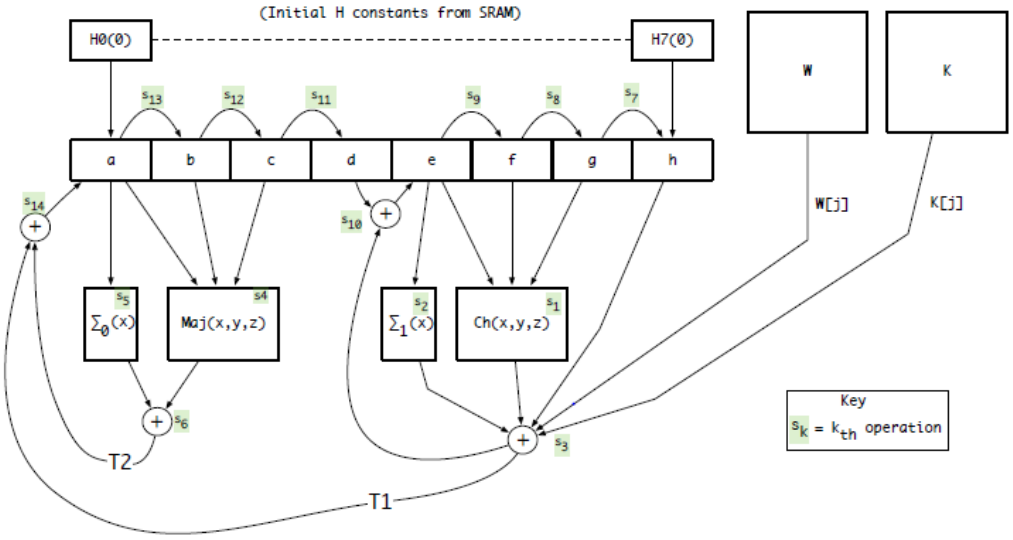
W[i] = fn(W[i-2],W[i-7],W[i-15],W[i-16]) where i=16,17,18….64



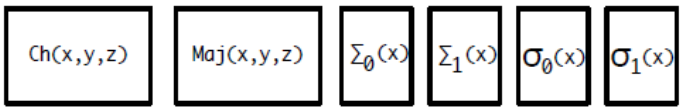
4) An array of 64 elements is constructed by reading from the K memory.

5) An array of 8 elements is created by reading from the H memory.

6) 8 32-bit registers a-h are initialized with elements of H memory.64 iterations are performed on the elements of these registers using the elements of K and W array. The following operations are performed in the following order :



The various operations shown in the diagrams, such as, Maj, Ch, σ , Σ , S, R are shown below.



$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\Sigma_0(x) = S^2(x) \oplus S^{13}(x) \oplus S^{22}(x)$$

$$\Sigma_1(x) = S^6(x) \oplus S^{11}(x) \oplus S^{25}(x)$$

$$\sigma_0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x)$$

$$\sigma_1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$$

$\oplus$	bitwise XOR
$\wedge$	bitwise AND
$\vee$	bitwise OR
$\neg$	bitwise complement
$+$	mod 2^{32} addition
R^n	right shift by n bits
S^n	right rotation by n bits

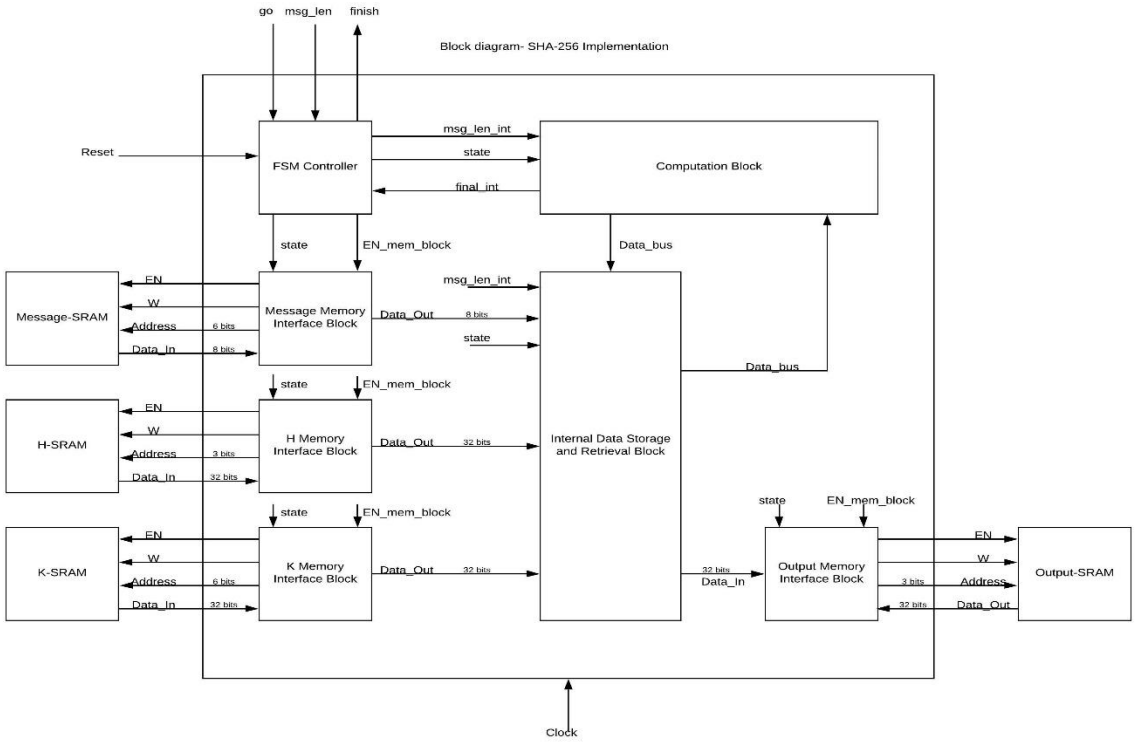
7) After completion of 64 iterations, the elements of H array are updated in the following manner:

The updated H vector represents the hash of the message.

Hash of message, M

8) The elements of the H vector is then written to an output memory.

High level architecture:

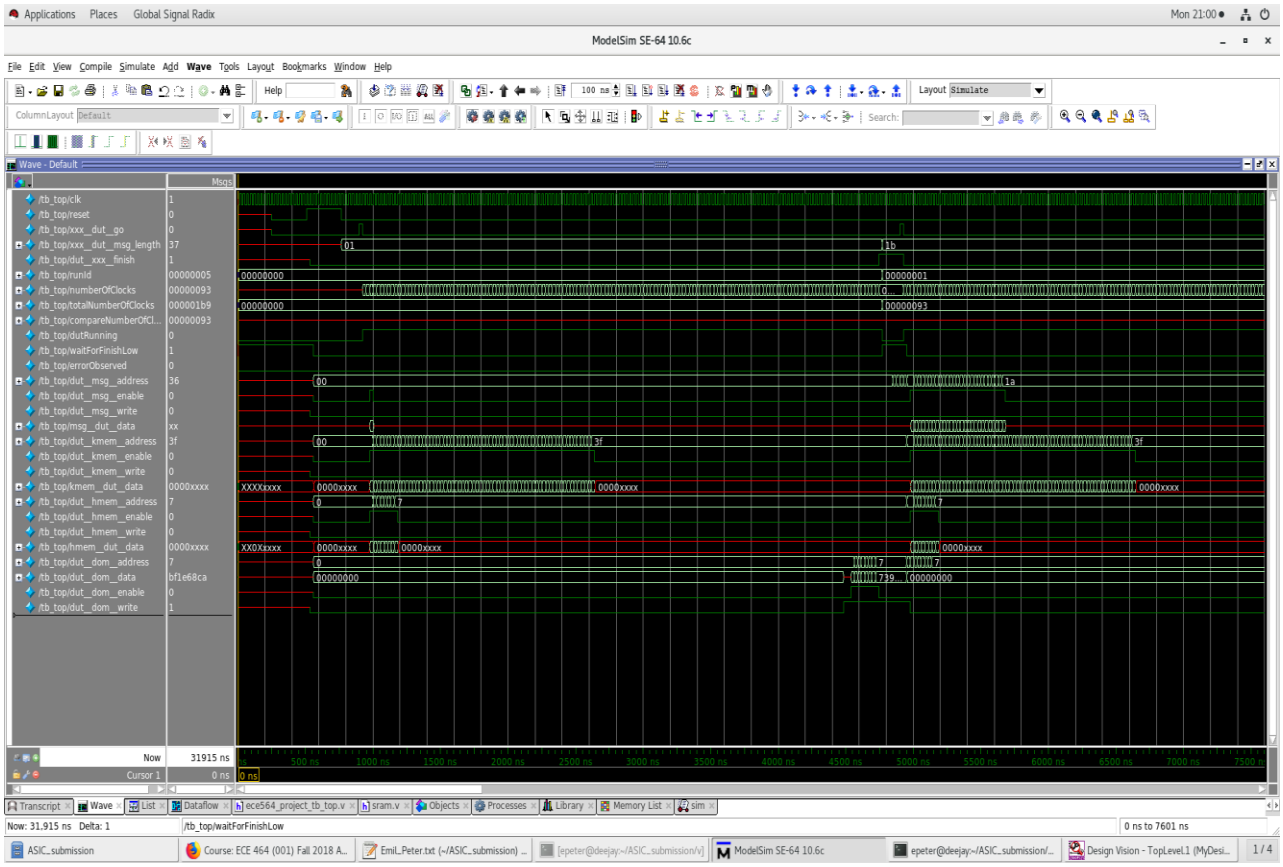


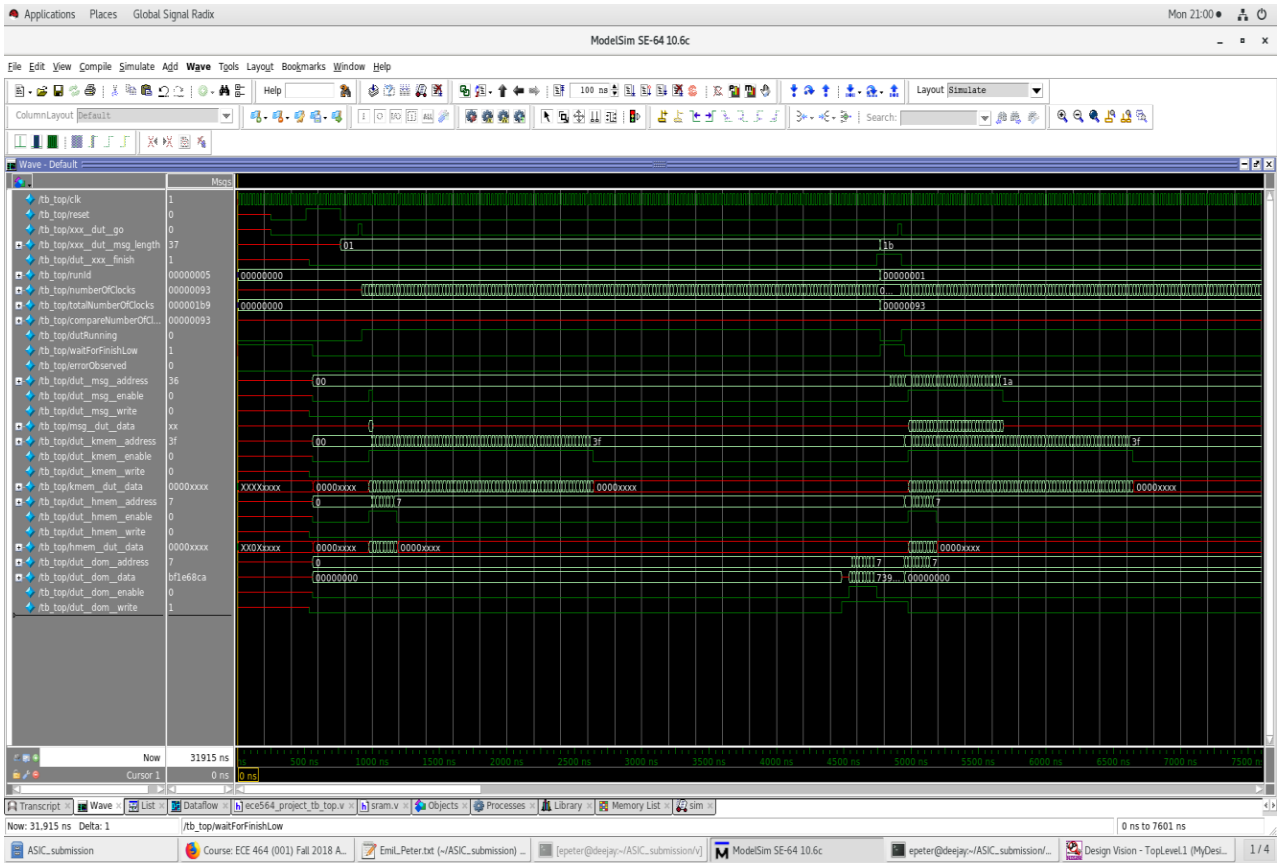
3. Interface Specification

Direction	Type	Width	Name	Function
Output	Reg		dut__xxx__finish	The signal goes high after final hash is written to the output memory.
Input	Wire		xxx__dut__go	The design starts its operation when go signal goes high
Input	Wire	[5:0]	xxx__dut__msg__length	This signal specifies the length of the input message.
			K Constant Memory	
Output	Reg	[5:0]	dut__kmem__address	This is the address of the element to be fetched from K memory.
Output	Reg		dut__kmem__enable	This signals the K memory for read operation
Output	Reg		dut__kmem__write	Signal is high for write operation and low for read.
Output	Reg	[31:0]	dut__kmem__data	This is the 32-bit data to be written to K memory.
Input	wire	[31:0]	kmem__dut__data	This is the 32-bit data read from K memory.
			H initial data memory	
Output	Reg	[8:0]	dut__hmem__address	This is the address of the element to be fetched from H memory.
Output	Reg		dut__hmem__enable	This signals the H memory for read operation
Output	Reg		dut__hmem__write	Signal is high for write operation and low for read.
Output	Reg	[31:0]	dut__hmem__data	This is the 32-bit data to be written to H memory.
Input	wire	[31:0]	hmem__dut__data	This is the 32-bit data read from H memory.
			Message memory	
Output	Reg	[8:0]	dut__msg__address	This is the address of the element to be fetched from message memory.
Output	Reg		dut__msg__enable	This signals the message memory for read operation
Output	Reg		dut__msg__write	Signal is high for write operation and low for read.

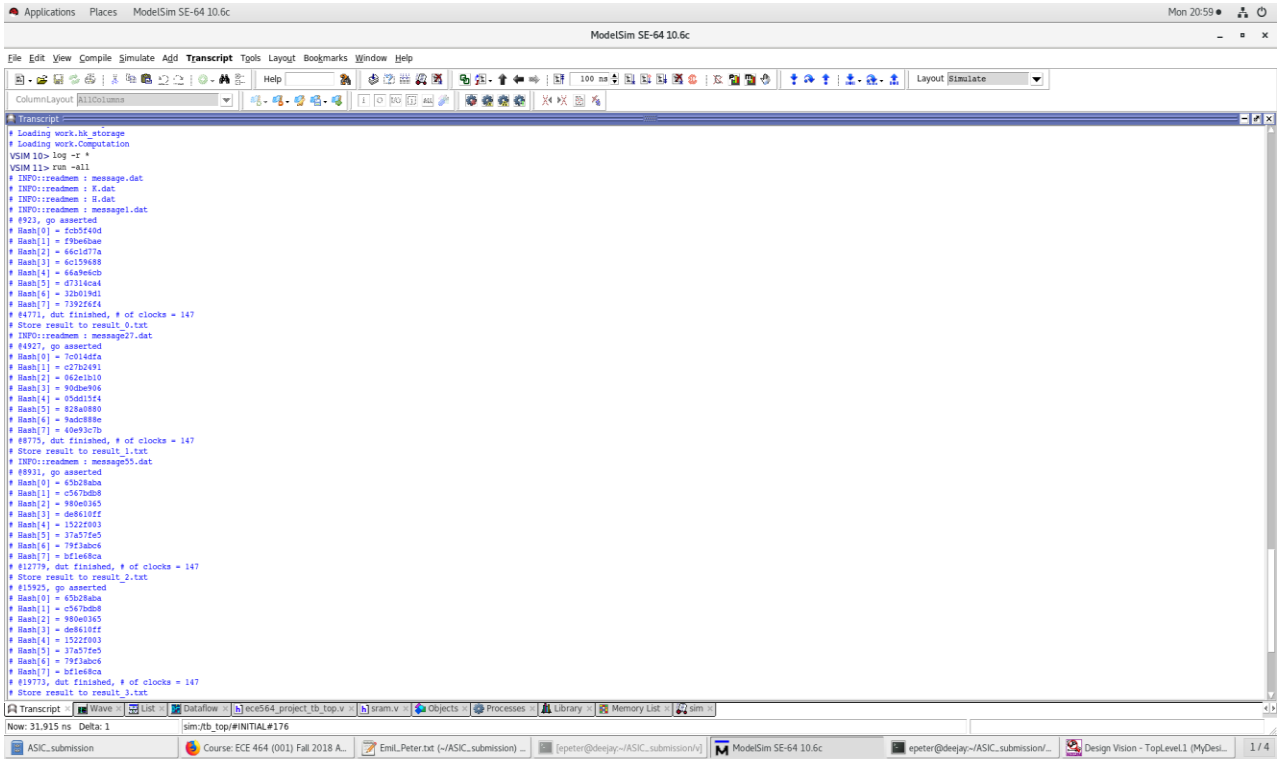
Output	Reg	[7:0]	dut__msg__data	This is the 32-bit data to be written to message memory.
Input	Wire	[7:0]	msg__dut__data	This is the 32-bit data read from message memory.
			Output data memory	
Output	Reg	[3:0]	dut__dom__address	This is the address to which the element is to be written in output memory.
Output	Reg		dut__dom__enable	This signals the output memory for read operation
Output	Reg		dut__dom__write	Signal is high for write operation and low for read.
Output	Reg	[31:0]	dut__dom__data	This is the 32-bit data to be written to output memory.
			General	
Input	Wire		clk	This is the clock signal to synchronize all operations.
Input	Wire		reset	It resets all registers in the design.

Timing diagram:

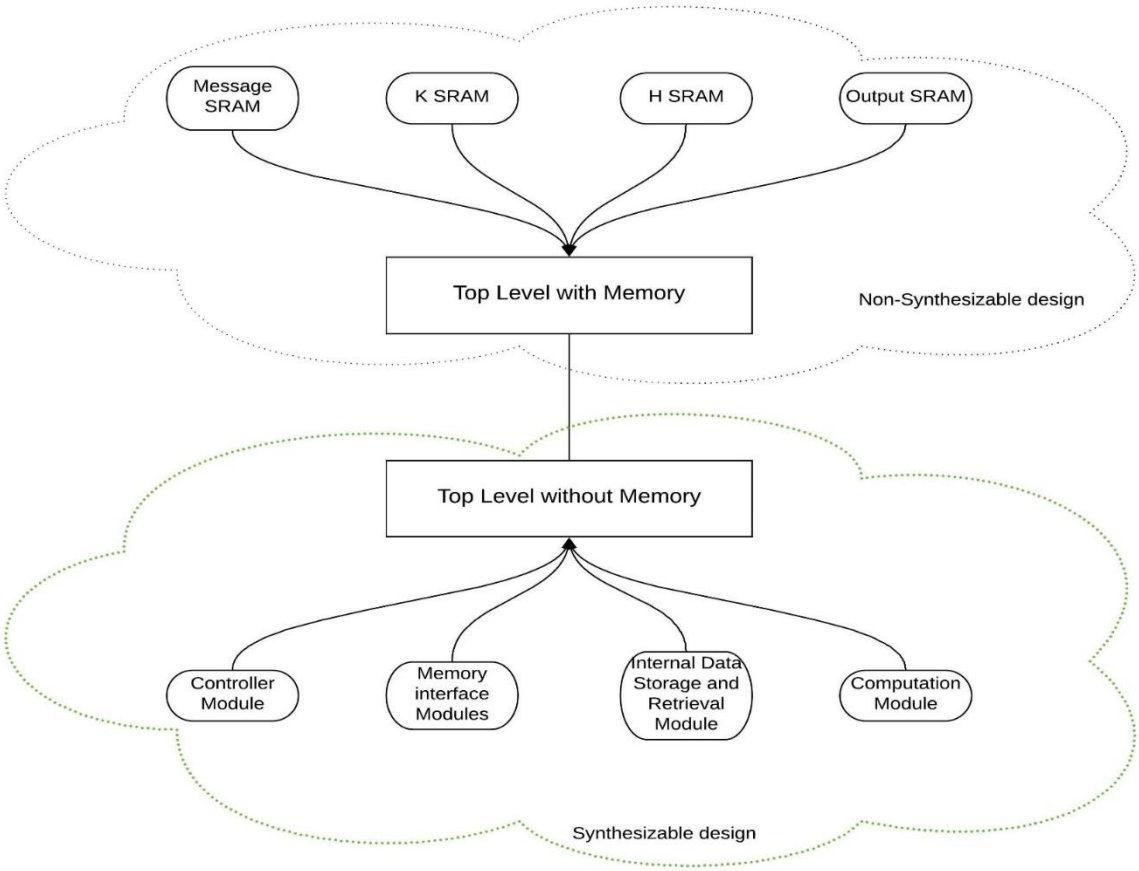




Output obtained:



4. Technical Implementation

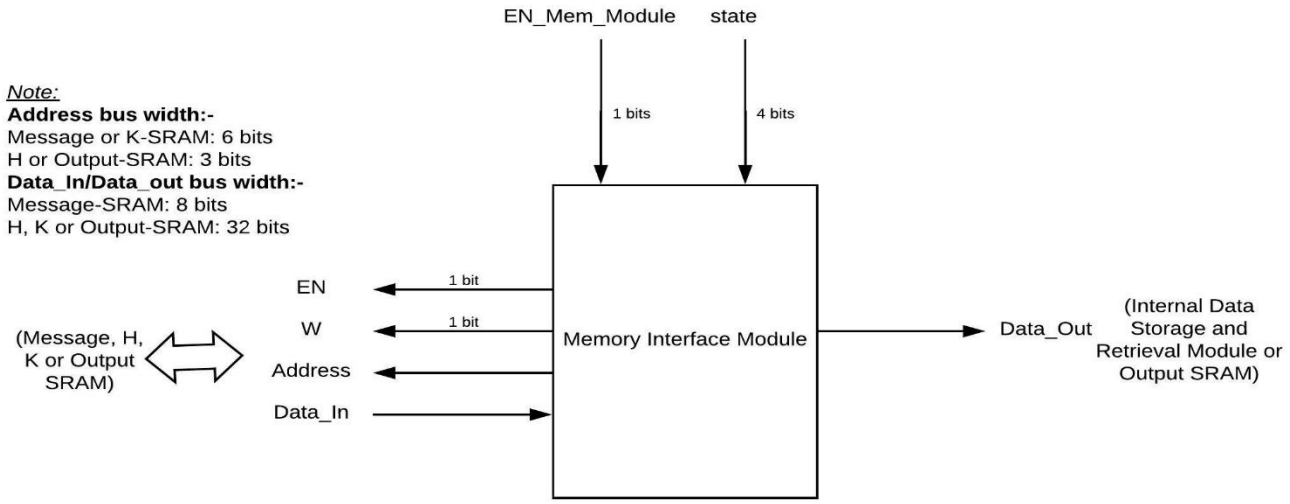


The design consists of two hierarchies:

1) Top level with memory:
   This level consists of the K, H, Message and output SRAMs to which/from data is to be written/read. This a non-synthesizable design.
2) Top level without memory:
   This level consists of Controller, memory interface, internal data storage and computation module. This is a synthesizable design.

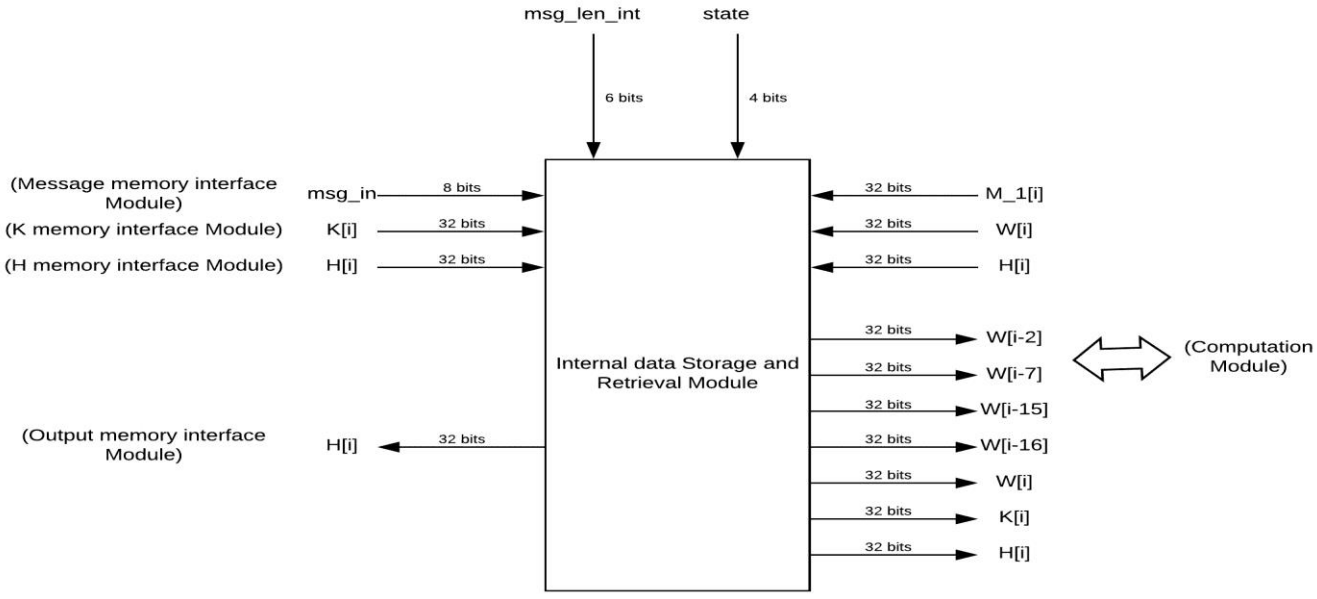
Modules in the design:

Memory Interface module-



The function of the module is to generate the address to be send to the SRAM based on the state and EN_Mem_module signals received from the controller. It also passes on the data received from the SRAM to the internal storage module.
Registers used: 1 8-bit register for data
1 32-bit register for address

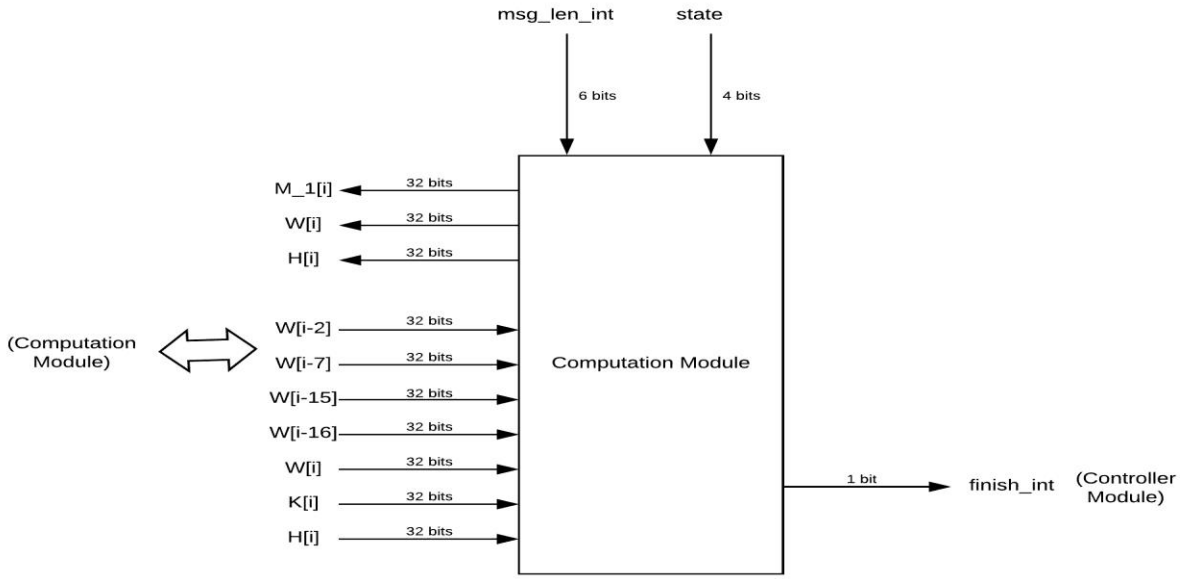
Internal data Storage and retrieval module:



The function of the module is to store the M_1 array, W array, K array and the H array. This module sends out the required data for computation to the computation block and also reads in the data send from the Memory interface module based on the state signal from the controller.

Registers used: 16 32-bit registers for M_1 array
64 32-bit registers for W array
64 32-bit registers for K array
8 32-bit registers for H array

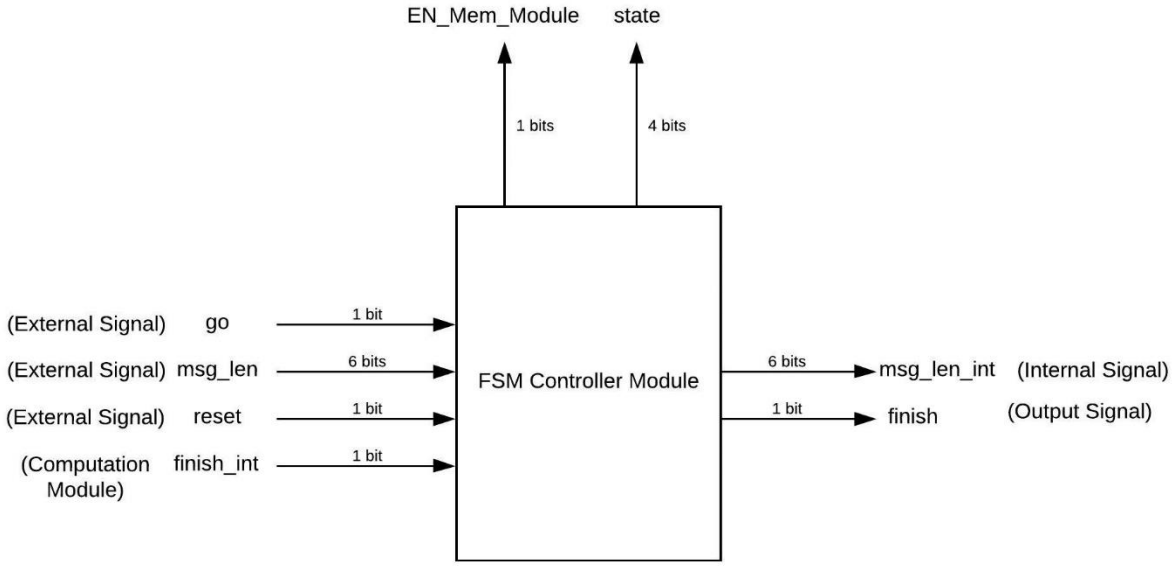
Computation module:



This module performs the computational operations like $\sigma_1(x)$, $\sigma_2(x)$, $\xi_1(x)$, $\xi_0(x)$, Maj (x.y,z) and Ch(x,y,z). This module is also responsible for reading in the message from the memory interface module and creating the M vector which is to be send to the Internal data Storage and retrieval module.

Registers used: 16 32-bit registers for M vector
8 32-bit registers for a-h elements
1 32-bit register for W(i) element

Controller module:



This module generates all the control signals for the design and reads in the external signals to the system like go and msg_len signals. It determines the state of the system and generates the finish signal on the completion of generating the hash of the message.
Registers used: 1 1-bit register for go signal
1 6-bit register for msg_len signal
1 1-bit register for finish signal

5. Verification

Each and every module was tested individually with self-designed test benches. Each module was then carefully integrated one by one in a step by step procedure, testing the top module thus formed after each integration. The design was tested with different message inputs of varying lengths ranging from 1-55 characters. The hardware was also tested whether it responded to go signals coming in between computations. Debugging was performed by comparing the output at each stage with the output generated by the python script for corresponding input message.

6. Results Achieved

Hardware was successfully designed with an area of 75283.0549 um^2, operating at clock period of 20ns and completing the operations in 441 clock cycles for the given test bench. Hardware also generated correct hash code for each input message of lengths varying from 1-55 characters. The generated hash code was successfully written to the output memory as well as to an output text file. The performance number obtained was 663.99×10^6 um^2 ns.

7. Conclusions

The hardware successfully generated correct hash code for each input message applied to it. Different length messages were applied to the design and tested for correct output. The design was also immune to any go signals applied while computation was carried out. The design achieved performance number of 663.99 x 10^6 um^2 ns.

APPENDIX

Design Code:

```
//------------------------------------------------------------------------
//------------------------------------------------------------------------
// DUT


`define MSG_LENGTH 5

//Top module
module MyDesign #(parameter OUTPUT_LENGTH      = 8,
                  parameter MAX_MESSAGE_LENGTH  = 55,
                  parameter NUMBER_OF_Ks        = 64,
                  parameter NUMBER_OF_Hs        = 8 ,
                  parameter SYMBOL_WIDTH        = 8,
                        parameter DATA_WIDTH1       = 32,
                        parameter DATA_WIDTH        = 8 )
        (

        //------------------------------------------------------------------------
        // Control
        //
        output reg                              dut__xxx__finish     ,
        input  wire                             xxx__dut__go         ,
        input  wire  [ $clog2(MAX_MESSAGE_LENGTH):0] xxx__dut__msg_length ,

        //------------------------------------------------------------------------
        // Message memory interface
        //
        output reg  [ $clog2(MAX_MESSAGE_LENGTH)-1:0]   dut__msg__address  ,  //
address of letter
        output reg                              dut__msg__enable   ,
        output reg                              dut__msg__write    ,
        input  wire [7:0]                         msg__dut__data     ,  // read each letter

        //------------------------------------------------------------------------
        // K memory interface
        //
        output reg  [ $clog2(NUMBER_OF_Ks)-1:0]     dut__kmem__address  ,
        output reg                            dut__kmem__enable   ,
        output reg                            dut__kmem__write    ,
        input  wire [31:0]                      kmem__dut__data     ,  // read data

        //------------------------------------------------------------------------
```

```
        // H memory interface
        //
        output reg  [ $clog2(NUMBER_OF_Hs)-1:0]     dut__hmem__address  ,
        output reg                          dut__hmem__enable   ,
        output reg                          dut__hmem__write    ,
        input  wire [31:0]                     hmem__dut__data     ,  // read data


        //------------------------------------------------------------------------
        // Output data memory
        //
        output reg  [ $clog2(OUTPUT_LENGTH)-1:0]    dut__dom__address  ,
        output reg  [31:0]                     dut__dom__data     ,  // write data
        output reg                          dut__dom__enable   ,
        output reg                          dut__dom__write    ,


        //------------------------------
        // General
        //
        input  wire             clk           ,
        input  wire             reset

        );

 //--------------------------------------------------------------------------
 //
 //<<<<----  YOUR CODE HERE    ---->>>>

 //`include "v564.vh"

wire [DATA_WIDTH-1:0] data_out;
wire [DATA_WIDTH1-1:0] data1;
wire [DATA_WIDTH1-1:0] data2;
wire [DATA_WIDTH1-1:0] Hash_out;
wire [31:0] H_out;
wire [31:0] K_out;
wire [31:0] W_out;
wire [5:0] count_hout;
wire [6:0] count_h;
wire [6:0] count_in;
wire [6:0] write_h;
wire [31:0] Hash_final;
wire reset_address_gen;
wire start_message_counter;
wire start_address_gen;
```

```
wire start_address_gen_write;
wire reset_address_gen_write;
wire reset_wgenerate_counters;
wire reset_counter;
wire [ $clog2(MAX_MESSAGE_LENGTH)-1:0] EndAddress_message;
wire [ $clog2(NUMBER_OF_Ks)-1:0] EndAddress_K;
wire [ $clog2(NUMBER_OF_Hs)-1:0] EndAddress_H;
wire [ $clog2(OUTPUT_LENGTH)-1:0] EndAddress_O;
wire [ $clog2(MAX_MESSAGE_LENGTH):0] internal_msg_length;
wire [5:0] count;
wire int_reset;
wire internal_finish;
wire kmem__write;
wire hmem__write;
wire msg__write;
wire dom__write;

Controller
#(.MAX_MESSAGE_LENGTH(MAX_MESSAGE_LENGTH),.NUMBER_OF_Ks(NU
MBER_OF_Ks),.NUMBER_OF_Hs(NUMBER_OF_Hs),.OUTPUT_LENGTH(OUTPU
T_LENGTH)) controller
(.clk(clk),.reset(reset),.int_reset(int_reset),.dut__xxx__finish(dut__xxx__finish),.xxx__du
t__go(xxx__dut__go),.xxx__dut__msg_length(xxx__dut__msg_length),.internal_msg_le
ngth(internal_msg_length),.internal_finish(internal_finish),.EndAddress_message(EndAd
dress_message),.EndAddress_K(EndAddress_K),.EndAddress_H(EndAddress_H),.EndA
ddress_O(EndAddress_O),.reset_address_gen(reset_address_gen),.start_address_gen(star
t_address_gen),.reset_counter(reset_counter),.reset_wgenerate_counters(reset_wgenerate
_counters),.start_message_counter(start_message_counter),.reset_address_gen_write(rese
t_address_gen_write),
.start_address_gen_write(start_address_gen_write),.count(count),.write_h(write_h),.dut_
_dom__write(dut__dom__write),.dut__hmem__write(dut__hmem__write),.dut__kmem_
_write(dut__kmem__write),.dut__msg__write(dut__msg__write),.kmem__write(kmem_
_write),.dom__write(dom__write),.hmem__write(hmem__write),.msg__write(msg__writ
e));
Memory_Interface_message
message_mem(.clk(clk),.reset(int_reset),.start1(reset_address_gen),.start2(start_address_
gen),.EndAddress(EndAddress_message),.address(dut__msg__address),.enable(dut__ms
g__enable),.data_in(msg__dut__data),.data_out(data_out));
Memory_Interface_K
K_mem(.clk(clk),.reset(int_reset),.start1(reset_address_gen),.start2(start_address_gen),.E
ndAddress(EndAddress_K),.enable(dut__kmem__enable),.address(dut__kmem__address
),.data_in(kmem__dut__data),.data_out(data1));
Memory_Interface_H
H_mem(.clk(clk),.reset(int_reset),.start1(reset_address_gen),.start2(start_address_gen),.E
ndAddress(EndAddress_H),.enable(dut__hmem__enable),.address(dut__hmem__address
),.data_in(hmem__dut__data),.data_out(data2));
```

```
Memory_Interface_O
output_mem(.clk(clk),.reset(int_reset),.start1(reset_address_gen_write),.start2(start_addr
ess_gen_write),.EndAddress(EndAddress_O),.address(dut__dom__address),.enable(dut_
_dom__enable),.data_in(Hash_out),.data_out(dut__dom__data));
Warray
w_vector(.clk(clk),.reset(int_reset),.start(start_message_counter),.start1(reset_wgenerate_
counters),.message(data_out),.message_length(internal_msg_length),.row_select(count_h
out),.W_out(W_out),.count(count),.msg__write(msg__write));
hk_storage
hk_storage(.clk(clk),.reset(int_reset),.data1(data1),.data2(data2),.start1(reset_counter),.H
_out(H_out),.K_out(K_out),.row_select(count_hout),.count_h(count_h),.Hash_final(Hash
_final),.count_in(count_in),.Hash_out(Hash_out),.internal_finish(internal_finish),.dom__
write(dom__write),.write_h(write_h),.hmem__write(hmem__write),.kmem__write(kmem
__write));
Computation
compute_block(.clk(clk),.reset(int_reset),.K_out(K_out),.W_out(W_out),.count_h1(count
_h),.count_h2(count_hout),.H_out(H_out),.start1(reset_address_gen),.Hash_final(Hash_fi
nal),.count_in(count_in));

  //
  //----------------------------------------------------------------------------

endmodule

//Controller module
module Controller #(parameter MAX_MESSAGE_LENGTH  = 55,parameter
NUMBER_OF_Ks        = 64,
                parameter NUMBER_OF_Hs        = 8 ,parameter OUTPUT_LENGTH       =
8 )

(clk,reset,int_reset,dut__xxx__finish,xxx__dut__go,xxx__dut__msg_length,internal_msg
_length,internal_finish,EndAddress_message,EndAddress_K,EndAddress_H,EndAddress
_O,reset_address_gen,start_address_gen,reset_counter,reset_wgenerate_counters,start_m
essage_counter,reset_address_gen_write,start_address_gen_write,count,write_h,kmem__
write,dom__write,msg__write,hmem__write,dut__dom__write,dut__hmem__write,dut__
kmem__write,dut__msg__write);
input clk,reset;
output dut__xxx__finish;
output int_reset;
input xxx__dut__go;
input internal_finish;
input [5:0] count;
input [ $clog2(MAX_MESSAGE_LENGTH):0] xxx__dut__msg_length;
output [ $clog2(MAX_MESSAGE_LENGTH):0] internal_msg_length;
output [ $clog2(MAX_MESSAGE_LENGTH)-1:0] EndAddress_message;
output [ $clog2(NUMBER_OF_Ks)-1:0] EndAddress_K;
```

```
output [ $clog2(NUMBER_OF_Hs)-1:0] EndAddress_H;
output [ $clog2(OUTPUT_LENGTH)-1:0] EndAddress_O;
output reset_address_gen;
output start_address_gen;
output reset_address_gen_write;
output start_address_gen_write;
output reset_counter;
output reset_wgenerate_counters;
output start_message_counter;
output dut__dom__write;
output dut__hmem__write;
output dut__kmem__write;
output dut__msg__write;
input kmem__write;
input hmem__write;
input dom__write;
input msg__write;
input [6:0] write_h;
reg [ $clog2(MAX_MESSAGE_LENGTH):0] internal_msg_length;
reg [ $clog2(MAX_MESSAGE_LENGTH)-1:0] EndAddress_message;
reg [ $clog2(NUMBER_OF_Ks)-1:0] EndAddress_K;
reg [ $clog2(NUMBER_OF_Hs)-1:0] EndAddress_H;
reg [ $clog2(OUTPUT_LENGTH)-1:0] EndAddress_O;
reg reset_address_gen;
reg start_address_gen;
reg reset_address_gen_write;
reg start_address_gen_write;
reg reset_counter;
reg reset_wgenerate_counters;
reg start_message_counter;
reg [2:0] counter;
reg internal_go;
reg dut__xxx__finish;
reg int_reset;
reg dut__dom__write;
reg dut__hmem__write;
reg dut__kmem__write;
reg dut__msg__write;
reg flag;
always@(posedge clk)
if (reset)
begin
        internal_msg_length<=0;
        EndAddress_message<=0;
        EndAddress_K<=0;
        EndAddress_H<=0;
```

```
        EndAddress_O<=0;
end
else
begin
        internal_msg_length<=xxx__dut__msg_length;
                //Generating an internal message signal
        EndAddress_message<=(internal_msg_length-1'h1);
                //Generating end addresses for accessing H,K,message and output memory
        EndAddress_K<=6'h3f;
        EndAddress_H<=6'h7;
        EndAddress_O<=6'h7;
end
always@(posedge clk)
                                //flag to prevent go signal to interrupt the design before
finishing computation
if (reset)
        flag<=0;
else if (xxx__dut__go)
        flag<=1'b1;
else if (dut__xxx__finish)
        flag<=0;
always@(posedge clk)
                                //Generating internal reset signal
if (reset)
        int_reset<=reset;
else if (xxx__dut__go && (!flag))
        int_reset<=xxx__dut__go;
else
        int_reset<=0;
always@(posedge clk)
                                //Generating internal go signal
if (reset)
        internal_go<=0;
else if(xxx__dut__go==1'b1 && (!flag))
        internal_go<=xxx__dut__go;
else if (flag)
        internal_go<=internal_go;

always@(posedge clk)
                                //Counter controller to generate signals to differnt modules
if (reset)
        counter<=3'b0;
else if(xxx__dut__go==1'b1 && (!flag))
        counter<=3'b0;
else if ((counter<3'b011 && internal_go==1'b1) || (counter!=3'b101 && write_h<7'ha
&& write_h>7'h0))
```

```
	counter<=counter+1'b1;
always@(posedge clk)
if (reset)
begin
	start_address_gen<=0;
	reset_address_gen<=0;
	reset_counter<=0;
	start_message_counter<=0;
	reset_wgenerate_counters<=0;
	start_address_gen_write<=0;
	reset_address_gen_write<=0;
end
else if (internal_go==1'b1)
begin
	case (counter)
	3'b000:  reset_address_gen<=1'b1;
		//Signal to reset address generating counter
	3'b001:  begin
			start_address_gen<=1'b1;
			//Signal to start address generating counter
			reset_address_gen<=1'b0;
			reset_counter<=1'b1;
			//Signal to reset counters in all modules
		end
	3'b010:  begin
			reset_wgenerate_counters<=1'b1;
			//Signal to reset counter generating W vector
			reset_counter<=1'b0;
			start_message_counter<=1'b1;
			//Signal to start counter to read in message
		end
	3'b011:  reset_wgenerate_counters<=1'b0;
	3'b100: reset_address_gen_write<=1'b1;
		//Signal to reset counter to generate address for output memory
	3'b101: begin
			start_address_gen_write<=1'b1;
			//Signal to start counter to generate address for output memory
			reset_address_gen_write<=1'b0;
			if (count==((internal_msg_length)+1))
				start_message_counter<=0;
			end
	endcase
end
always@(posedge clk)
				//Generating finish signal based on internal finish signal
if (reset)
```

```
        dut__xxx__finish<=0;
else if( xxx__dut__go==1'b1 && (!flag))
        dut__xxx__finish<=0;
else if (int_reset)
        dut__xxx__finish<=0;
else if (internal_finish==1'b1)
        dut__xxx__finish<=1'b1;
always@(posedge clk)
                                //Generating write signals for memories
if (reset)
begin
        dut__msg__write<=0;
        dut__kmem__write<=0;
        dut__hmem__write<=0;
        dut__dom__write<=0;
end
else
begin
        dut__msg__write<=msg__write;
        dut__kmem__write<=kmem__write;
        dut__hmem__write<=hmem__write;
        dut__dom__write<=dom__write;
end
endmodule

//Memory interface module for H
module Memory_Interface_H
(clk,reset,start1,start2,EndAddress,enable,address,data_in,data_out);
parameter NUMBER_OF_Hs= 8;
parameter  ADDR_WIDTH= $clog2(NUMBER_OF_Hs );
parameter  DATA_WIDTH= 32;
input reset,clk;
input [ADDR_WIDTH-1:0]   EndAddress;
input [DATA_WIDTH-1:0]data_in;
input start1,start2;
output enable;
output [DATA_WIDTH-1:0] data_out;
output [ADDR_WIDTH-1:0] address;
reg [ADDR_WIDTH-1:0] address;
reg [DATA_WIDTH-1:0] data_out;
wire complete;
reg enable;
assign complete=(address==EndAddress);
        //Checking if the address has reached end address
```

```
always@(posedge clk)
		//Counter generating address for reading from H memory
if (reset)
	begin
		address<=6'h0;
		enable<=0;
	end
else if (start1)
	begin
		address<=6'h0;
		enable<=1;
	end
else if (!complete && start2)
	begin
		address<=address+1'b1;
	end
else if (complete)
		enable<=0;


always@(posedge clk)
		//Reading in and storing data read from memory
if (reset)
	data_out<=32'b0;
else
	data_out<=data_in;
endmodule

module Memory_Interface_K
(clk,reset,start1,start2,EndAddress,enable,address,data_in,data_out);
parameter NUMBER_OF_Ks= 64;
parameter  ADDR_WIDTH= $clog2(NUMBER_OF_Ks );
parameter  DATA_WIDTH= 32;
input reset,clk;
input [ADDR_WIDTH-1:0]   EndAddress;
input [DATA_WIDTH-1:0]data_in;
input start1,start2;
output enable;
output [DATA_WIDTH-1:0] data_out;
output [ADDR_WIDTH-1:0] address;
reg [ADDR_WIDTH-1:0] address;
reg [DATA_WIDTH-1:0] data_out;
wire complete;
reg enable;
assign complete=(address==EndAddress);
	//Checking if the address has reached end address
```

```
always@(posedge clk)
                //Counter generating address for reading from K memory
if (reset)
        begin
                address<=6'h0;
                enable<=0;
        end
else if (start1)
        begin
                address<=6'h0;
                enable<=1;
        end
else if (!complete && start2)
        begin
                address<=address+1'b1;
        end
else if (complete)
                enable<=0;


always@(posedge clk)
                //Reading in and storing data read from memory
if (reset)
        data_out<=32'b0;
else
        data_out<=data_in;
endmodule

module Memory_Interface_message
(clk,reset,start1,start2,EndAddress,enable,address,data_in,data_out);
parameter MAX_MESSAGE_LENGTH  = 55;
parameter  ADDR_WIDTH= $clog2(MAX_MESSAGE_LENGTH);
parameter  DATA_WIDTH= 8;
input reset,clk;
input [ADDR_WIDTH-1:0]   EndAddress;
input [DATA_WIDTH-1:0]data_in;
input start1,start2;
output enable;
output [DATA_WIDTH-1:0] data_out;
output [ADDR_WIDTH-1:0] address;
reg [ADDR_WIDTH-1:0] address;
reg [DATA_WIDTH-1:0] data_out;
wire complete;
reg enable;
```

```
assign complete=(address==EndAddress);
        //Checking if the address has reached end address

always@(posedge clk)
                //Counter generating address for reading from message memory
if (reset)
        begin
                address<=6'h0;
                enable<=0;
        end
else if (start1)
        begin
                address<=6'h0;
                enable<=1;
        end
else if (!complete && start2)
        begin
                address<=address+1'b1;
        end
else if (complete)
                enable<=0;


always@(posedge clk)
                //Reading in and storing data read from memory
if (reset)
        data_out<=32'b0;
else
        data_out<=data_in;
endmodule

module Memory_Interface_O
(clk,reset,start1,start2,EndAddress,enable,address,data_in,data_out);
parameter OUTPUT_LENGTH= 8;
parameter  ADDR_WIDTH= $clog2(OUTPUT_LENGTH);
parameter  DATA_WIDTH= 32;
input reset,clk;
input [ADDR_WIDTH-1:0]   EndAddress;
input [DATA_WIDTH-1:0]data_in;
input start1,start2;
output enable;
output [DATA_WIDTH-1:0] data_out;
output [ADDR_WIDTH-1:0] address;
reg [ADDR_WIDTH-1:0] address;
reg [DATA_WIDTH-1:0] data_out;
wire complete;
```

```
reg enable;
assign complete=(address==EndAddress);
	//Checking if the address has reached end address

always@(posedge clk)
	//Counter generating address for writing to output memory
if (reset)
	begin
		address<=6'h0;
		enable<=0;
	end
else if (start1)
	begin
		address<=6'h0;
		enable<=1;
	end
else if (!complete && start2)
	begin
		address<=address+1'b1;
	end
else if (complete)
		enable<=0;


always@(posedge clk)
	//Writing data to output memory
if (reset)
	data_out<=32'b0;
else
	data_out<=data_in;
endmodule

//Warray module to read in message and generate W array
module
Warray(clk,reset,start,start1,message,message_length,row_select,W_out,count,msg__writ
e);
parameter a = 8'b10000000;
parameter MAX_MESSAGE_LENGTH  = 55;
integer i,j,z;
input clk,reset,start,start1;
input [7:0] message;
input [ $clog2(MAX_MESSAGE_LENGTH):0] message_length;
output [5:0] row_select;
output [31:0] W_out;
output [5:0] count;
output msg__write;
```

```
reg [5:0] count;
reg [1:0] column_select;
reg [5:0] row_select;
reg [7:0] data_in;
reg [31:0] W[63:0];
reg [6:0] count_wout;
reg [31:0] W_out;
wire [8:0] length;
reg msg__write;
assign length=(message_length<<3);
        //Converting the message length into binary
always@(posedge clk)
        //Counter to read in message from message memory
begin
if (reset )
begin
        count<=0;
        msg__write<=1'b0;
        //write signal to message memory set to 0
end
else if (start && count!=(message_length+1))
begin
        count<=count+1'b1;
        msg__write<=1'b0;
end
end
always@(posedge clk)
        //Reading and storing message from message memory
begin
if (reset)
begin
        data_in<=0;
end
else if (count==message_length)
        data_in<=a;
else if (start && count!=(message_length+1))
        data_in<=message;
else if (count==(message_length+1))
begin
        data_in<=0;
end
end
always@(posedge clk)
        //Column decoder for storing message in W array
if (reset)
        column_select<=0;
```

```
else if(start1)
	column_select<=0;
else
	column_select<=column_select+1'b1;
always@(posedge clk)
	//Row decoder for storing message in W array
if (reset)
	row_select<=0;
else if (start1)
	row_select<=0;
else if (column_select==2'b11 && row_select!=6'he )
	row_select<=row_select+1'b1;
always@(posedge clk)
	//Creating W array
if (reset)
begin
for (i=0;i<64;i=i+1)begin
	W[i]<=0;
end
end
else if (row_select==6'he)
	//Creating W[16]-W[64]
begin
	for(j=0;j<16;j=j+1)
	begin
	W[z]<=W[z];
	end
	for(j=16;j<64;j=j+1)
	begin
		W[j]<=((({W[j-2],W[j-2]}>>17)^({W[j-2],W[j-2]}>>19))^(W[j-
2]>>10))+W[j-7]+W[j-16]+((({W[j-15],W[j-15]}>>7)^({W[j-15],W[j-15]}>>18))^(W[j-
15]>>3));
	end
end
else
		//Filling in message to create W[0]-W[15]
begin
	case(column_select)
		2'b00:if(reset)
			W[row_select][31:24]<=0;
		    else
			begin
			W[row_select][31:24]<=data_in;
			end
		2'b01:if(reset)
			W[row_select][23:16]<=0;
```

```
                    else
                        W[row_select][23:16]<=data_in;
                2'b10:if(reset)
                        W[row_select][15:8]<=0;
                    else
                        W[row_select][15:8]<=data_in;
                2'b11:if(reset)
                        W[row_select][7:0]<=0;
                    else
                        W[row_select][7:0]<=data_in;
        endcase
        W[15][7:0]<=length[7:0];
        //Appending 1 to the end after filling in message
        W[15][15:8]<={{7{1'b0}},length[8]};
end
always@(posedge clk)
        //Counter to output elements of W array for computation
if (reset)
        count_wout<=6'h0;
else if(start1)
        count_wout<=6'h0;
else if (row_select==6'he && count_wout!=7'h40)
        count_wout<=count_wout+1'b1;
always@(posedge clk)
        //Outputting elements of W array
if (reset)
        W_out<=0;
else if (row_select!=6'he)
        W_out<=0;
else if (count_wout!=7'h40)
        W_out<=W[count_wout];
endmodule

//Module for performing 64 iterations on a-h registers
module
Computation(clk,reset,K_out,W_out,count_h1,count_h2,H_out,start1,Hash_final,count_i
n);
integer i;
input clk,reset,start1;
input [31:0] K_out;
input [31:0] W_out;
input [31:0] H_out;
input [6:0] count_h1;
input [5:0] count_h2;
output [31:0] Hash_final;
output [6:0] count_in;
```

```
reg [31:0] H[7:0];
reg [6:0] count_h;
reg [6:0] count_in;
reg [6:0] h_index;
reg [31:0] K;
reg [31:0] W;
reg [31:0] Hash_final;
wire [31:0] Ch;
wire [31:0] Maj;
wire [31:0] T1;
wire [31:0] T2;
wire [31:0] E1;
wire [31:0] E0;
wire [31:0] temp1;
wire [31:0] temp2;
wire [31:0] temp3;
reg [1:0] count_for_compute;
always@(posedge clk)
        //Counter to read in elements of H from hk_storage module
if (reset)
        count_h<=6'h0;
else if (start1)
        count_h<=6'h0;
else if (count_h1==7'h8 && count_h!=7'h0a)
        count_h<=count_h+1'b1;
always@(posedge clk)
        //Counter for counting the number of iterations
if (reset)
        count_in<=6'h0;
else if (start1)
        count_in<=6'h0;
else if (count_h2==6'he && count_in!=7'h41)
        count_in<=count_in+1'b1;
always@(posedge clk)
        //Counter for synchronizing the reading in of H elements and starting of
computation
if (reset)
        count_for_compute<=6'h0;
else if(start1)
        count_for_compute<=6'h0;
else if (count_h2==6'he && count_for_compute!=2'h1)
        count_for_compute<=count_for_compute+1'b1;
always@(posedge clk)
if (reset)
begin
        K<=0;
```

```
        W<=0;
end
else
begin
K<=K_out;
W<=W_out;
end
//Functions for computation
assign Ch=(H[4]&H[5])^((~H[4])&H[6]);
assign E1=(({H[4],H[4]}>>6)^({H[4],H[4]}>>11))^({H[4],H[4]}>>25);
assign temp1=Ch+E1;
assign temp2=temp1+H[7];
assign temp3=K_out+W_out;
assign T1=temp2+temp3;
assign Maj=(H[0]&H[1])^(H[0]&H[2])^(H[1]&H[2]);
assign E0=(({H[0],H[0]}>>2)^({H[0],H[0]}>>13))^({H[0],H[0]}>>22);
assign T2=Maj+E0;
always@(posedge clk)
        //Iterations performed on a-h registers
if (reset)
begin
for (i=0;i<8;i=i+1)begin
        H[i]<=0;
end
end
else if (count_in!=7'h41 && count_for_compute==2'h1)
begin
H[7]<=H[6];
H[6]<=H[5];
H[5]<=H[4];
H[4]<=H[3]+T1;
H[3]<=H[2];
H[2]<=H[1];
H[1]<=H[0];
H[0]<=T1+T2;
end
else if (count_h1==7'h8 && count_h!=7'h9)
        H[count_h-1'b1]<=H_out;
always@(posedge clk)
        //Counter to generate index for outputting final result after 64 iterations
if (reset)
        h_index<=0;
else if (start1)
        h_index<=0;
else if (count_in==7'h41 && h_index!=7'h8)
        h_index<=h_index+1'b1;
```

```
always@(posedge clk)
        //Outputting final results
if (reset)
        Hash_final<=0;
else if (count_in!=7'h41)
        Hash_final<=0;
else if (h_index!=7'h8)
        Hash_final<=H[h_index];
endmodule

//Module for reading in H and K elements
module
hk_storage(clk,reset,data1,data2,start1,H_out,K_out,row_select,count_h,Hash_final,Hash
_out,count_in,internal_finish,dom__write,write_h,hmem__write,kmem__write);
integer x,y,z;
input clk,reset,start1;
input [31:0] data1;
input [31:0] data2;
output [31:0] H_out;
output [31:0] K_out;
output [31:0] Hash_out;
output internal_finish;
input [31:0] Hash_final;
input [5:0] row_select;
output [6:0] count_h;
input [6:0] count_in;
output dom__write;
output hmem__write;
output kmem__write;
output [6:0] write_h;
reg [31:0] K[63:0];
reg [31:0] H[7:0];
reg [31:0] computed_h[7:0];
reg [31:0] H_out;
reg [31:0] K_out;
reg [31:0] Hash_out;
reg [6:0] count_k;
reg [6:0] count_h;
reg [6:0] count_hout;
reg [6:0] count_kout;
reg [6:0] final_count;
reg [6:0] write_h;
reg flag;
reg internal_finish;
reg dom__write;
reg hmem__write;
```

```
reg kmem__write;


always@(posedge clk)
        //Counter for reading in K elements from K memory
if (reset)
begin
        count_k<=6'h0;
        kmem__write<=1'b0;
end
else if(start1)
begin
        count_k<=6'h0;
        kmem__write<=1'b0;
end
else if (count_k!=7'h40)
begin
        count_k<=count_k+1'b1;
        kmem__write<=1'b0;
end
always@(posedge clk)
        //Counter for reading in H elements from H memory
if (reset)
begin
        count_h<=6'h0;
        hmem__write<=1'b0;
end
else if(start1)
begin
        count_h<=6'h0;
        hmem__write<=1'b0;
end
else if (count_h!=7'h8)
begin
        count_h<=count_h+1'b1;
        hmem__write<=1'b0;
end
always@(posedge clk)
        //Outputting H elements for computation
if (reset)
        count_hout<=6'h0;
else if(start1)
        count_hout<=6'h0;
else if (count_hout!=7'h8 && count_h==7'h8)
        count_hout<=count_hout+1'b1;
```

```
always@(posedge clk)
	//Outputting K elements for computation
if (reset)
	count_kout<=6'h0;
else if (start1)
	count_kout<=6'h0;
else if (row_select==6'he && count_kout!=7'h40)
	count_kout<=count_kout+1'b1;
always@(posedge clk)
	//Counter for reading in results after 64 iterations
if (reset)
begin
	final_count<=6'h0;
end
else if (start1)
begin
	final_count<=6'h0;
end
else if (final_count!=7'h9 && count_in==7'h41)
	final_count<=final_count+1'b1;
always@(posedge clk)
	//Storing elements of K in K array
if (reset)
begin
for (x=0;x<64;x=x+1)begin
	K[x]<=0;
end
end
else
	K[count_k]<=data1;
always@(posedge clk)
	//Outputting elements of H array
if (reset)
	H_out<=0;
else if (count_h!=7'h8)
	H_out<=0;
else if (count_hout!=7'h8)
	H_out<=H[count_hout];
always@(posedge clk)
	//Outputting elements of K array
if (reset)
	K_out<=0;
else if (row_select!=6'he)
	K_out<=0;
else if (count_kout!=7'h40)
	K_out<=K[count_kout];
```

```
always@(posedge clk)
	//Storing the results after 64 iterations
if (reset)
begin
for (z=0;z<8;z=z+1)begin
	computed_h[z]<=0;
end
end
else if (final_count!=7'h9 && count_in==7'h41)
	computed_h[final_count-1'h1]<=Hash_final;
always@(posedge clk)
	//Computation of final hash for given message
if (reset)
begin
	for (y=0;y<8;y=y+1)
	begin
	H[y]<=0;
	end
	flag<=0;
end
else if(start1)
	flag<=0;
else if (final_count==7'h9 && flag!=1'b1)
begin
H[0]<=computed_h[0]+H[0];
H[1]<=computed_h[1]+H[1];
H[2]<=computed_h[2]+H[2];
H[3]<=computed_h[3]+H[3];
H[4]<=computed_h[4]+H[4];
H[5]<=computed_h[5]+H[5];
H[6]<=computed_h[6]+H[6];
H[7]<=computed_h[7]+H[7];
flag<=1'b1;
end
else
begin
	H[count_h]<=data2;
end
always@(posedge clk)
	//Counter for outputting final hash to output memory
if (reset)
	write_h<=6'h0;
else if (start1)
	write_h<=6'h0;
else if (flag==1'b1 && write_h!=7'ha)
	write_h<=write_h+1'b1;
```

```
always@(posedge clk)
	//Outputting final hash to output memory
if (reset)
begin
	Hash_out<=0;
	dom__write<=0;
end
else if (write_h!=7'ha && flag==1'b1)
begin
	Hash_out<=H[write_h-2'b10];
	dom__write<=1'b1;
end
always@(posedge clk)
	//Setting internal finish high after outputting final hash
if (reset)
	internal_finish<=0;
else if (write_h==7'ha)
	internal_finish<=1'b1;
endmodule
```

Testbench:

```
//------------------------------------------------------------------------------------------------
// To run simulation
//
// vlog -sv ece564_project_tb_top.v
// vsim -c -do "run 1us; quit" tb_top
//
// you can display the expected intermediate and output results by adding defines to vlog
// vlog -sv +define+TB_DISPLAY_INTERMEDIATE+TB_DISPLAY_EXPECTED
ece564_project_tb_top.v
//
//
// ECE464 and EOL students
//`define ECE464
// All students
// - set based on your message memory depth
`define MSG_LENGTH 5

// synopsys translate_off
/*
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW_fp_cmp.v"
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW_fp_cmp_DG.v"
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW_fp_mult.v"
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW_fp_mac.v"
```

```
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW_fp_dp2.v"
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW_fp_ifp_conv.v"
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW_ifp_fp_conv.v"
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW_ifp_mult.v"
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW_ifp_addsub.v"
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW02_mult_5_stage.v"
`include "/afs/eos.ncsu.edu/dist/synopsys2013/syn/dw/sim_ver/DW01_addsub.v"
*/

//synopsys translate_on
//
//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
// Tesbench
// - instantiate the DUT and testbench

module tb_top ();


  parameter CLK_PHASE=12.5  ;
  `ifdef ECE464
  parameter OUTPUT_LENGTH      = 16 ;
  `else
  parameter OUTPUT_LENGTH      = 8 ;
  `endif
  parameter MAX_MESSAGE_LENGTH  = 55;
  parameter NUMBER_OF_Ks       = 64;
  parameter NUMBER_OF_Hs       = 8 ;
  parameter SYMBOL_WIDTH       = 8 ;

  //--------------------------------------------------------------------------
  // General
  //
  reg                           clk               ;
  reg                           reset             ;
  reg                           xxx__dut__go        ;
  reg   [ $clog2(MAX_MESSAGE_LENGTH):0] xxx__dut__msg_length ;
  wire                           dut__xxx__finish     ;

  integer runId ;
  integer numberOfClocks ;
  integer totalNumberOfClocks ;
  integer compareNumberOfClocks ;
  // compare files
  reg [31:0] comparisonMem [OUTPUT_LENGTH-1:0];
  reg [31:0] resultMem [OUTPUT_LENGTH-1:0];
```

```
  bit dutRunning, waitForFinishLow;
  bit errorObserved ;
  string outputType;

  initial begin
     `ifdef ECE464
     outputType = "M_1";
     `else
     outputType = "Hash";
     `endif
   end


  wire [ $clog2(MAX_MESSAGE_LENGTH)-1:0]   dut__msg__address  ;  // address of
letter
  wire                          dut__msg__enable   ;
  wire                          dut__msg__write    ;
  wire [7:0]                      msg__dut__data     ;  // read each letter

  wire [ $clog2(NUMBER_OF_Ks)-1:0]        dut__kmem__address  ;
  wire                          dut__kmem__enable   ;
  wire                          dut__kmem__write    ;
  wire [31:0]                     kmem__dut__data     ;  // read data

  wire [ $clog2(NUMBER_OF_Hs )-1:0]       dut__hmem__address  ;
  wire                          dut__hmem__enable   ;
  wire                          dut__hmem__write    ;
  wire [31:0]                     hmem__dut__data     ;  // read data

  wire [ $clog2(OUTPUT_LENGTH)-1:0]       dut__dom__address  ;
  wire [31:0]                     dut__dom__data     ;  // write data
  wire                          dut__dom__enable   ;
  wire                          dut__dom__write    ;

  //------------------------------------------------------------------------
  //------------------------------------------------------------------------
  //------------------------------------------------------------------------

  sram  #(.ADDR_WIDTH    ($clog2(MAX_MESSAGE_LENGTH)),
         .DATA_WIDTH    (SYMBOL_WIDTH             ),
         .MEM_INIT_FILE ("message.dat"            ))
        msg_mem  (
         .address      ( dut__msg__address  ),
         .write_data   ( {SYMBOL_WIDTH {1'b0}}),
         .read_data    ( msg__dut__data     ),
```

```
        .enable       ( dut__msg__enable   ),
        .write        ( dut__msg__write    ),
        .clock        ( clk                )
       );

sram  #(.ADDR_WIDTH    ( $clog2(NUMBER_OF_Ks)),
        .DATA_WIDTH    (32),
        .MEM_INIT_FILE ("K.dat"         ))
        kmem_mem (

        .address     ( dut__kmem__address  ),
        .write_data  ( 'd0                 ),
        .read_data   ( kmem__dut__data     ),
        .enable      ( dut__kmem__enable   ),
        .write       ( dut__kmem__write    ),

        .clock       ( clk                 )
       );

sram  #(.ADDR_WIDTH    ( $clog2(NUMBER_OF_Hs )),
        .DATA_WIDTH    (32),
        .MEM_INIT_FILE ("H.dat"         ))
        hmem_mem (

        .address     ( dut__hmem__address  ),
        .write_data  ( 'd0                 ),
        .read_data   ( hmem__dut__data     ),
        .enable      ( dut__hmem__enable   ),
        .write       ( dut__hmem__write    ),

        .clock       ( clk                 )
       );

sram  #(.ADDR_WIDTH  ($clog2(OUTPUT_LENGTH)),
        .DATA_WIDTH  (32))
       dom_mem  (

        .address     ( dut__dom__address  ),
        .write_data  ( dut__dom__data     ),
        .read_data   (                    ),
        .enable      ( dut__dom__enable   ),
        .write       ( dut__dom__write    ),

        .clock       ( clk                )
       );
```

```
//------------------------------------------------------------------------
//------------------------------------------------------------------------
//------------------------------------------------------------------------
//-----------------------------
// Testbench
//

//------------------------------------------------------------------------
//  clk
initial
  begin
     clk                  = 1'b0;
     forever # CLK_PHASE clk = ~clk;
  end

initial
  begin
     errorObserved = 0;
     dutRunning = 0;
     waitForFinishLow = 1;
     totalNumberOfClocks = 0;
     repeat(10) @(posedge clk);
     reset = 0;
     xxx__dut__go = 0;
     repeat(10) @(posedge clk);
     reset = 1;
     repeat(10) @(posedge clk);
     reset = 0;
     xxx__dut__msg_length = 1;
     msg_mem.memFile = $sformatf("message1.dat");
     ->msg_mem.loadMemory;
     repeat(5) @(posedge clk);
     xxx__dut__go = 1;
     repeat(1) @(posedge clk);
     xxx__dut__go = 0;
     wait(!dut__xxx__finish);
     wait(dut__xxx__finish);
     repeat(1) @(posedge clk);
     totalNumberOfClocks = totalNumberOfClocks + numberOfClocks;

     msg_mem.memFile = $sformatf("message27.dat");
     ->msg_mem.loadMemory;
     xxx__dut__msg_length = 27;
     repeat(5) @(posedge clk);
     xxx__dut__go = 1;
```

```
repeat(1) @(posedge clk);
xxx__dut__go = 0;
wait(!dut__xxx__finish);
wait(dut__xxx__finish);
repeat(1) @(posedge clk);
totalNumberOfClocks = totalNumberOfClocks + numberOfClocks;

msg_mem.memFile = $sformatf("message55.dat");
->msg_mem.loadMemory;
xxx__dut__msg_length = 55;
repeat(5) @(posedge clk);
xxx__dut__go = 1;
repeat(1) @(posedge clk);
xxx__dut__go = 0;
wait(!dut__xxx__finish);
wait(dut__xxx__finish);
repeat(1) @(posedge clk);
totalNumberOfClocks = totalNumberOfClocks + numberOfClocks;

repeat(100) @(posedge clk);

reset = 1;
repeat(10) @(posedge clk);
reset = 0;
repeat(10) @(posedge clk);
xxx__dut__go = 1;
repeat(1) @(posedge clk);
xxx__dut__go = 0;
wait(!dut__xxx__finish);
wait(dut__xxx__finish);
compareNumberOfClocks = numberOfClocks;

repeat(100) @(posedge clk);

reset = 1;
repeat(10) @(posedge clk);
reset = 0;
repeat(10) @(posedge clk);
xxx__dut__go = 1;
repeat(1) @(posedge clk);
xxx__dut__go = 0;
wait(!dut__xxx__finish);

// message length of 55 ensures time should be >= 55
// generate an additional go after 30 cycles
repeat(30) @(posedge clk);
```

```
        $display("@%00t, Extra go during processing", $time);
        xxx__dut__go = 1;
        repeat(2) @(posedge clk);
        xxx__dut__go = 0;

        wait(dut__xxx__finish);

        repeat(100) @(posedge clk);

        // make sure extra go doesnt change behavior
        if (compareNumberOfClocks != numberOfClocks) begin
          $display("@%00t, ERROR:Extra go caused change in number of clocks :
expected %0d, got %0d", $time, compareNumberOfClocks, numberOfClocks);
          errorObserved = 1;
        end
        else begin
          $display("@%00t, Extra go didnt change behavior", $time);
        end

        repeat(100) @(posedge clk);

        $readmemh("result_0.txt", resultMem);
        `ifdef ECE464
        $readmemh("message1_result_ece464.txt", comparisonMem);
        `else
        $readmemh("message1_result.txt", comparisonMem);
        `endif
        for (int i=0; i<OUTPUT_LENGTH; i=i+1) begin
          if (comparisonMem[i] != resultMem[i]) begin
            $display("@%00t, ERROR:Message length 1, result content mismatch at
address %0h, expected %8h, got %8h", $time, i, comparisonMem[i], resultMem[i]);
            errorObserved = 1;
          end
        end

        $readmemh("result_1.txt", resultMem);
        `ifdef ECE464
        $readmemh("message27_result_ece464.txt", comparisonMem);
        `else
        $readmemh("message27_result.txt", comparisonMem);
        `endif
        for (int i=0; i<OUTPUT_LENGTH; i=i+1) begin
          if (comparisonMem[i] != resultMem[i]) begin
            $display("@%00t, ERROR:Message length 27, result content mismatch at
address %0h, expected %8h, got %8h", $time, i, comparisonMem[i], resultMem[i]);
            errorObserved = 1;
```

```
        end
      end

      $readmemh("result_2.txt", resultMem);
      `ifdef ECE464
      $readmemh("message55_result_ece464.txt", comparisonMem);
      `else
      $readmemh("message55_result.txt", comparisonMem);
      `endif
      for (int i=0; i<OUTPUT_LENGTH; i=i+1) begin
        if (comparisonMem[i] != resultMem[i]) begin
          $display("@%00t, ERROR:Message length 55, result content mismatch at
address %0h, expected %8h, got %8h", $time, i, comparisonMem[i], resultMem[i]);
          errorObserved = 1;
        end
      end

      $display("@%00t, INFO:The total number of clocks for reporting is %0d", $time,
totalNumberOfClocks );

      if (errorObserved) begin
        $display("******************");
        $display("***** ERROR *****");
        $display("******************");
      end

      $finish;
    end

  always
    begin
      @(posedge clk);
      if ((xxx__dut__go == 1'b1) && ~dutRunning)
        begin
          $display("@%0t, go asserted", $time);
          numberOfClocks = 0;
          dutRunning = 1;
        end
      else if ((dut__xxx__finish == 1'b0) && waitForFinishLow)
        begin
          waitForFinishLow = 0;
          numberOfClocks = numberOfClocks + 1;
        end
      else if ((dut__xxx__finish == 1'b1) && ~waitForFinishLow && dutRunning)
        begin
          $display("@%00t, dut finished, # of clocks = %0d", $time, numberOfClocks);
```

```
        dutRunning = 0;
        waitForFinishLow = 1;
      end
    else if (dutRunning)
      begin
        numberOfClocks = numberOfClocks + 1;
      end
  end


 always
  begin
    @(posedge clk);
    if (dut__dom__enable == 1'b1)
      begin
        $display("%s[%0d] = %h", outputType, dut__dom__address, dut__dom__data);
      end
  end

 initial begin
    runId = 0;
  end

 always
  begin
    @(posedge clk);
    if (dut__xxx__finish == 1'b1)
      begin
        $display("Store result to result_%0d.txt", runId);
        $writememh($sformatf("result_%0d.txt", runId),dom_mem.mem);
        runId = runId+1;
      end
    wait (dut__xxx__finish == 1'b0);

  end


 //------------------------------------------------------------------------
 //------------------------------------------------------------------------
 // Stimulus

 //------------------------------------------------------------------------



//------------------------------------------------------------------------
```

```
//------------------------------------------------------------------------
// DUT
  MyDesign #(.OUTPUT_LENGTH      (OUTPUT_LENGTH     ),
             .MAX_MESSAGE_LENGTH (MAX_MESSAGE_LENGTH),
             .NUMBER_OF_Ks       (NUMBER_OF_Ks      ),
             .NUMBER_OF_Hs       (NUMBER_OF_Hs      ),
             .SYMBOL_WIDTH       (SYMBOL_WIDTH      ))

            dut(

          .xxx__dut__msg_length ( xxx__dut__msg_length ),
          .dut__xxx__finish     ( dut__xxx__finish     ),
          .xxx__dut__go         ( xxx__dut__go         ),
          .dut__msg__address    ( dut__msg__address    ),
          .msg__dut__data       ( msg__dut__data       ),
          .dut__msg__enable     ( dut__msg__enable     ),
          .dut__msg__write      ( dut__msg__write      ),

          .dut__kmem__address   ( dut__kmem__address   ),
          .kmem__dut__data      ( kmem__dut__data      ),
          .dut__kmem__enable    ( dut__kmem__enable    ),
          .dut__kmem__write     ( dut__kmem__write     ),

          .dut__hmem__address   ( dut__hmem__address   ),
          .hmem__dut__data      ( hmem__dut__data      ),
          .dut__hmem__enable    ( dut__hmem__enable    ),
          .dut__hmem__write     ( dut__hmem__write     ),

          .dut__dom__address    ( dut__dom__address    ),
          .dut__dom__data       ( dut__dom__data       ),
          .dut__dom__enable     ( dut__dom__enable     ),
          .dut__dom__write      ( dut__dom__write      ),

          .reset                ( reset                ),
          .clk                  ( clk                  )
         );



Endmodule
```

Timing report for setup violations:

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : MyDesign
Version: K-2015.06-SP1
Date   : Mon Nov 26 21:22:53 2018
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: slow   Library:
NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm
Wire Load Model Mode: top

  Startpoint: compute_block/H_reg[7][0]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: compute_block/H_reg[0][31]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Point                                       Incr       Path
  ---------------------------------------------------------------------------
  clock clk (rise edge)                     0.0000     0.0000
  clock network delay (ideal)               0.0000     0.0000
  compute_block/H_reg[7][0]/CK (DFF_X2)     0.0000 #   0.0000 r
  compute_block/H_reg[7][0]/Q (DFF_X2)      0.5987     0.5987 f
  add_3_root_add_0_root_compute_block/add_642/A[0] (MyDesign_DW01_add_12)
                                            0.0000     0.5987 f
  add_3_root_add_0_root_compute_block/add_642/U1/ZN (AND2_X4)
                                            0.2544     0.8531 f
  add_3_root_add_0_root_compute_block/add_642/U1_1/CO (FA_X1)
                                            0.4821     1.3352 f
  add_3_root_add_0_root_compute_block/add_642/U1_2/CO (FA_X1)
                                            0.5172     1.8524 f
  add_3_root_add_0_root_compute_block/add_642/U1_3/CO (FA_X1)
                                            0.5172     2.3697 f
  add_3_root_add_0_root_compute_block/add_642/U1_4/CO (FA_X1)
                                            0.5172     2.8869 f
  add_3_root_add_0_root_compute_block/add_642/U1_5/CO (FA_X1)
                                            0.5172     3.4041 f
```

add_3_root_add_0_root_compute_block/add_642/U1_6/CO (FA_X1)
0.5172 3.9214 f
add_3_root_add_0_root_compute_block/add_642/U1_7/CO (FA_X1)
0.5172 4.4386 f
add_3_root_add_0_root_compute_block/add_642/U1_8/CO (FA_X1)
0.5172 4.9559 f
add_3_root_add_0_root_compute_block/add_642/U1_9/CO (FA_X1)
0.5172 5.4731 f
add_3_root_add_0_root_compute_block/add_642/U1_10/CO (FA_X1)
0.5172 5.9904 f
add_3_root_add_0_root_compute_block/add_642/U1_11/CO (FA_X1)
0.5172 6.5076 f
add_3_root_add_0_root_compute_block/add_642/U1_12/CO (FA_X1)
0.5172 7.0249 f
add_3_root_add_0_root_compute_block/add_642/U1_13/S (FA_X1)
0.8374 7.8623 r
add_3_root_add_0_root_compute_block/add_642/SUM[13]
(MyDesign_DW01_add_12)
0.0000 7.8623 r
add_1_root_add_0_root_compute_block/add_642/B[13] (MyDesign_DW01_add_11)
0.0000 7.8623 r
add_1_root_add_0_root_compute_block/add_642/U1_13/S (FA_X1)
0.7063 8.5686 f
add_1_root_add_0_root_compute_block/add_642/SUM[13]
(MyDesign_DW01_add_11)
0.0000 8.5686 f
U23811/ZN (INV_X2) 0.1138 8.6823 r
U23812/ZN (INV_X4) 0.0435 8.7259 f
add_0_root_add_0_root_compute_block/add_642/B[13] (MyDesign_DW01_add_9)
0.0000 8.7259 f
add_0_root_add_0_root_compute_block/add_642/U1_13/CO (FA_X1)
0.5914 9.3173 f
add_0_root_add_0_root_compute_block/add_642/U1_14/CO (FA_X1)
0.5172 9.8346 f
add_0_root_add_0_root_compute_block/add_642/U1_15/CO (FA_X1)
0.5172 10.3518 f
add_0_root_add_0_root_compute_block/add_642/U1_16/CO (FA_X1)
0.5172 10.8690 f
add_0_root_add_0_root_compute_block/add_642/U1_17/CO (FA_X1)
0.5172 11.3863 f
add_0_root_add_0_root_compute_block/add_642/U1_18/CO (FA_X1)
0.5172 11.9035 f
add_0_root_add_0_root_compute_block/add_642/U1_19/CO (FA_X1)
0.5172 12.4208 f
add_0_root_add_0_root_compute_block/add_642/U1_20/CO (FA_X1)
0.5172 12.9380 f

```
  add_0_root_add_0_root_compute_block/add_642/U1_21/S (FA_X1)
                                          0.8034    13.7414 f
  add_0_root_add_0_root_compute_block/add_642/SUM[21] (MyDesign_DW01_add_9)
                                          0.0000    13.7414 f
  add_0_root_add_0_root_compute_block/add_662/B[21] (MyDesign_DW01_add_157)
                                          0.0000    13.7414 f
  add_0_root_add_0_root_compute_block/add_662/U1_21/CO (FA_X1)
                                          0.6764    14.4178 f
  add_0_root_add_0_root_compute_block/add_662/U1_22/CO (FA_X1)
                                          0.5172    14.9351 f
  add_0_root_add_0_root_compute_block/add_662/U1_23/CO (FA_X1)
                                          0.5172    15.4523 f
  add_0_root_add_0_root_compute_block/add_662/U1_24/CO (FA_X1)
                                          0.5172    15.9696 f
  add_0_root_add_0_root_compute_block/add_662/U1_25/CO (FA_X1)
                                          0.5172    16.4868 f
  add_0_root_add_0_root_compute_block/add_662/U1_26/CO (FA_X1)
                                          0.5172    17.0040 f
  add_0_root_add_0_root_compute_block/add_662/U1_27/CO (FA_X1)
                                          0.5377    17.5418 f
  add_0_root_add_0_root_compute_block/add_662/U5/ZN (NAND2_X1)
                                          0.2175    17.7593 r
  add_0_root_add_0_root_compute_block/add_662/U6/ZN (NAND3_X2)
                                          0.1250    17.8843 f
  add_0_root_add_0_root_compute_block/add_662/U11/ZN (NAND2_X1)
                                          0.1775    18.0618 r
  add_0_root_add_0_root_compute_block/add_662/U12/ZN (NAND3_X2)
                                          0.1173    18.1791 f
  add_0_root_add_0_root_compute_block/add_662/U1_30/CO (FA_X1)
                                          0.4889    18.6680 f
  add_0_root_add_0_root_compute_block/add_662/U1_31/S (FA_X1)
                                          0.7176    19.3856 r
  add_0_root_add_0_root_compute_block/add_662/SUM[31]
(MyDesign_DW01_add_157)
                                          0.0000    19.3856 r
  U23836/ZN (NAND2_X1)                    0.0769    19.4625 f
  U16669/ZN (NAND3_X2)                    0.1877    19.6502 r
  compute_block/H_reg[0][31]/D (DFF_X1)   0.0000    19.6502 r
  data arrival time                                 19.6502

  clock clk (rise edge)                  20.0000    20.0000
  clock network delay (ideal)             0.0000    20.0000
  clock uncertainty                      -0.0500    19.9500
  compute_block/H_reg[0][31]/CK (DFF_X1)  0.0000    19.9500 r
  library setup time                     -0.2008    19.7492
  data required time                                19.7492
```

```
  ---------------------------------------------------------------------------
  data required time                                 19.7492
  data arrival time                                 -19.6502
  ---------------------------------------------------------------------------
  slack (MET)                                        0.0990


1
```

Timing report for hold violations:

```
Information: Updating design information... (UID-85)
Warning: Design 'MyDesign' contains 1 high-fanout nets. A fanout number of 1000 will
be used for delay calculations involving these nets. (TIM-134)

****************************************
Report : timing
        -path full
        -delay min
        -max_paths 1
Design : MyDesign
Version: K-2015.06-SP1
Date   : Mon Nov 26 21:22:29 2018
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: fast   Library:
NangateOpenCellLibrary_PDKv1_2_v2008_10_fast_nldm
Wire Load Model Mode: top

  Startpoint: w_vector/column_select_reg[0]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: w_vector/column_select_reg[0]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: min

  Point                                    Incr       Path
  ---------------------------------------------------------------------------
  clock clk (rise edge)                    0.0000     0.0000
  clock network delay (ideal)              0.0000     0.0000
  w_vector/column_select_reg[0]/CK (DFF_X2)           0.0000 #   0.0000 r
  w_vector/column_select_reg[0]/Q (DFF_X2)            0.0592     0.0592 r
  U2091/ZN (NOR3_X2)                       0.0171     0.0763 f
```

w_vector/column_select_reg[0]/D (DFF_X2)	0.0000	0.0763 f	
data arrival time	0.0763		
clock clk (rise edge)	0.0000	0.0000	
clock network delay (ideal)	0.0000	0.0000	
clock uncertainty	0.0500	0.0500	
w_vector/column_select_reg[0]/CK (DFF_X2)	0.0000	0.0500 r	
library hold time	-0.0002	0.0498	
data required time	0.0498		
---	---	---	---
data required time	0.0498		
data arrival time	-0.0763		
---	---	---	---
slack (MET)	0.0265		