**ECE564 – ASIC and FPGA Design with Verilog**

/10

**Name:** Soumil Krishnanand Heble

**UnityID:** sheble

**StudentID:** 200207079

**Project Plan**

**Schedule:**

**10/24 –** load\_msg\_make\_m

**10/26 –** create\_w

**10/27 –** load\_kmem

**10/29 –** load\_hash

**10/31 –** Integration

**11/2 –** Verification

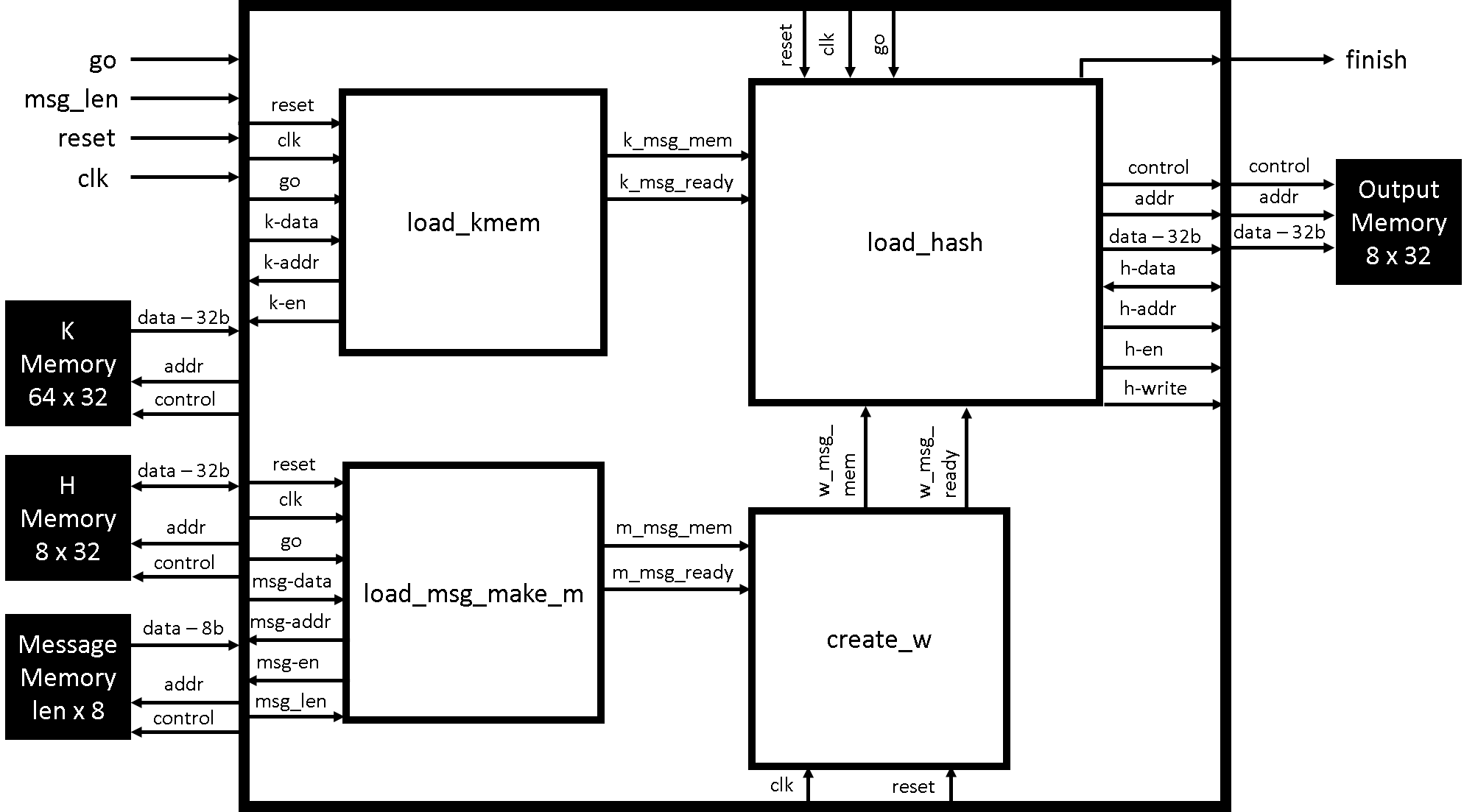
**Summary Risk Plan:**

\* I am unsure of the type of FSM I need to use whether master/slave or one within each of the modules and then use go/start flag from each preceding module that triggers the FSM in the next module.

\* I am unsure of how to improve the performance of the final hash module since the hash calculation algorithm will probably be the longest combinational logic chain.

**Brief Description of Mode of operation, including selected algorithms:** The shift and rotate operations being of a fixed value will be implemented by bit reordering. Modulo 32-bit adder used will be a carry look ahead adder for performance purposes. Parallelize the H-memory loading, K-memory loading along with the creation of M\_1 blocks + W array creation then start the hash calculation and finally end with writing the output to the memory. As of now I am considering to use internal FSM for each module triggered by go/start signals from the preceding modules.

**High Level Sketch: Top-level module = compute\_ sha256**



**Individual Module Description(s):**

* The outer box in the block diagram is the top level module with other modules listed below instantiated within the top level module. Each module will have an internal FSM that will be controlling the sequence of operations to be performed.

**Module:** load\_msg\_make\_mem

**Input(s):** reset, clk, xxx\_\_dut\_\_go, xxx\_\_dut\_\_msg\_length, msg\_\_dut\_\_data

**Output(s):** dut\_\_msg\_\_address, dut\_\_msg\_\_enable, m\_msg\_mem, m\_msg\_ready

**Register(s):** 64 x 8-bit m\_mem register file

**Brief Algorithm:** When go is asserted this module will clear the m\_mem register file then copy msg\_length bytes (ASCII characters) from the message memory to the m\_mem register file using a counter. Then right shift the msg\_length by 3 and store it in the last two bytes of the m\_mem register file to complete the preparation of M\_1 then assert the m\_msg\_ready flag.

**Module:** create\_w

**Input(s):** reset, clk, m\_msg\_mem, m\_msg\_ready

**Output(s):** w\_msg\_mem, w\_msg\_ready

**Register(s):** 64 x 32-bit w\_mem register file

**Brief Algorithm:** When m\_msg\_ready is asserted this module will clear the w\_mem register file then copy the contents of m\_mem (M\_1) from the m\_mem register file to the first 16 locations of the w\_mem register file using a counter. Then it will start computing the remaining locations of the w\_mem register file based on the SHA256 algorithm. After completion of the computation the w\_msg\_ready flag will be asserted.

**Module:** load\_kmem

**Input(s):** reset, clk, xxx\_\_dut\_\_go, kmem\_\_dut\_\_data

**Output(s):** dut\_\_kmem\_\_address, dut\_\_kmem\_\_enable, k\_msg\_mem, k\_msg\_ready

**Register(s):** 64 x 32-bit k\_mem register file

**Brief Algorithm:** When go is asserted this module will clear the k\_mem register file then copy the 64 elements from the K memory to the k\_mem register file using a counter. After completion of the copy it will then assert the k\_msg\_ready flag.

**Module:** load\_hash

**Input(s):** reset, clk, xxx\_\_dut\_\_go, hmem\_\_dut\_\_data, k\_msg\_mem, k\_msg\_ready, w\_msg\_mem, w\_msg\_ready

**Output(s):** dut\_\_hmem\_\_address, dut\_\_hmem\_\_enable, dut\_\_hmem\_\_write, dut\_\_hmem\_\_data, dut\_\_xxx\_\_finish

**Register(s):** 8 x 32-bit h\_mem register file

**Brief Algorithm:** When go is asserted this module will clear the h\_mem register file then copy the 8 elements from the H memory to the h\_mem register file using a counter. After completion it will wait for the k\_msg\_ready and w\_msg\_ready signals to become asserted. Then it will run the SHA256 sequence of computations for 64 iterations to generate the hash. Upon completion the the H values will be XOR’ed with the h\_mem register file and then written back to the H memory and then to the output memory. Finally the finished flag will be asserted.

**Behavioural Code:**

* if(go)
  + clear m\_mem, h\_mem, w\_mem and k\_mem
* for i=0 to length of message
  + copy byte from message memory to M\_1 array/register file
* right shift message length and store the 8 LSB to the last M\_1 array location and the remaining bits to the M\_1 array second last location
* for i=0 to 15
  + copy the half word from m\_mem to w\_mem array
* for i=16 to 63
  + compute the following

