ECE564 – ASIC and FPGA Design with Verilog

Project Plan

/10

Name: Soumil Krishnanand Heble

UnityID: sheble

StudentID: 200207079

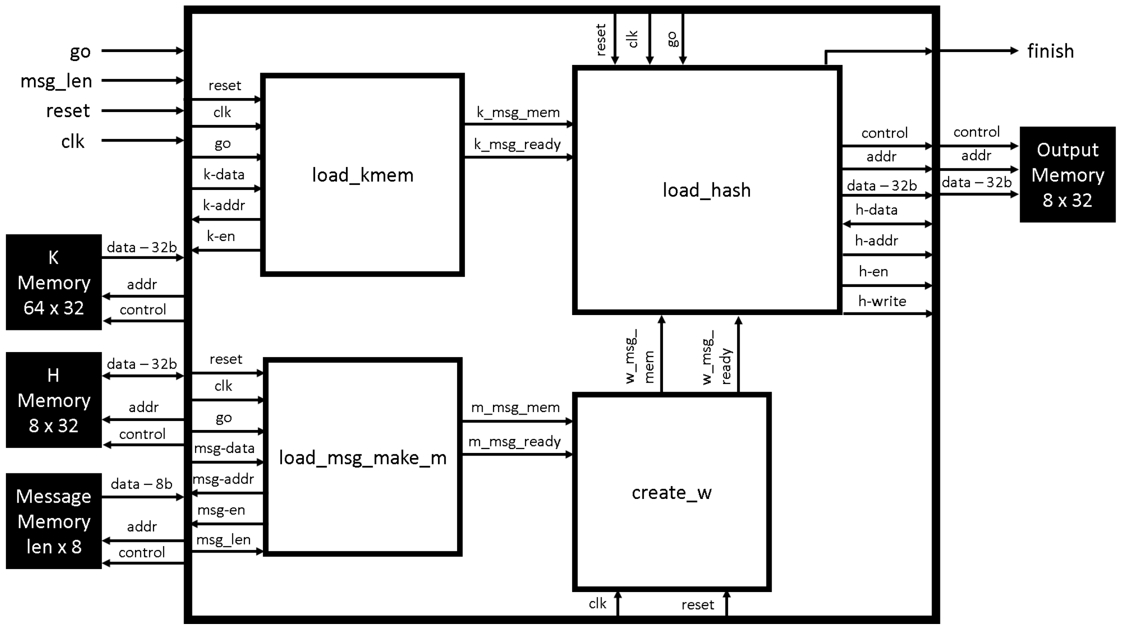
Summary Risk Plan:
* I am unsure of the type of FSM I need to use whether master/slave or one within each of the modules and then use go/start flag from each preceding module that triggers the FSM in the next module.

* 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.

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

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):

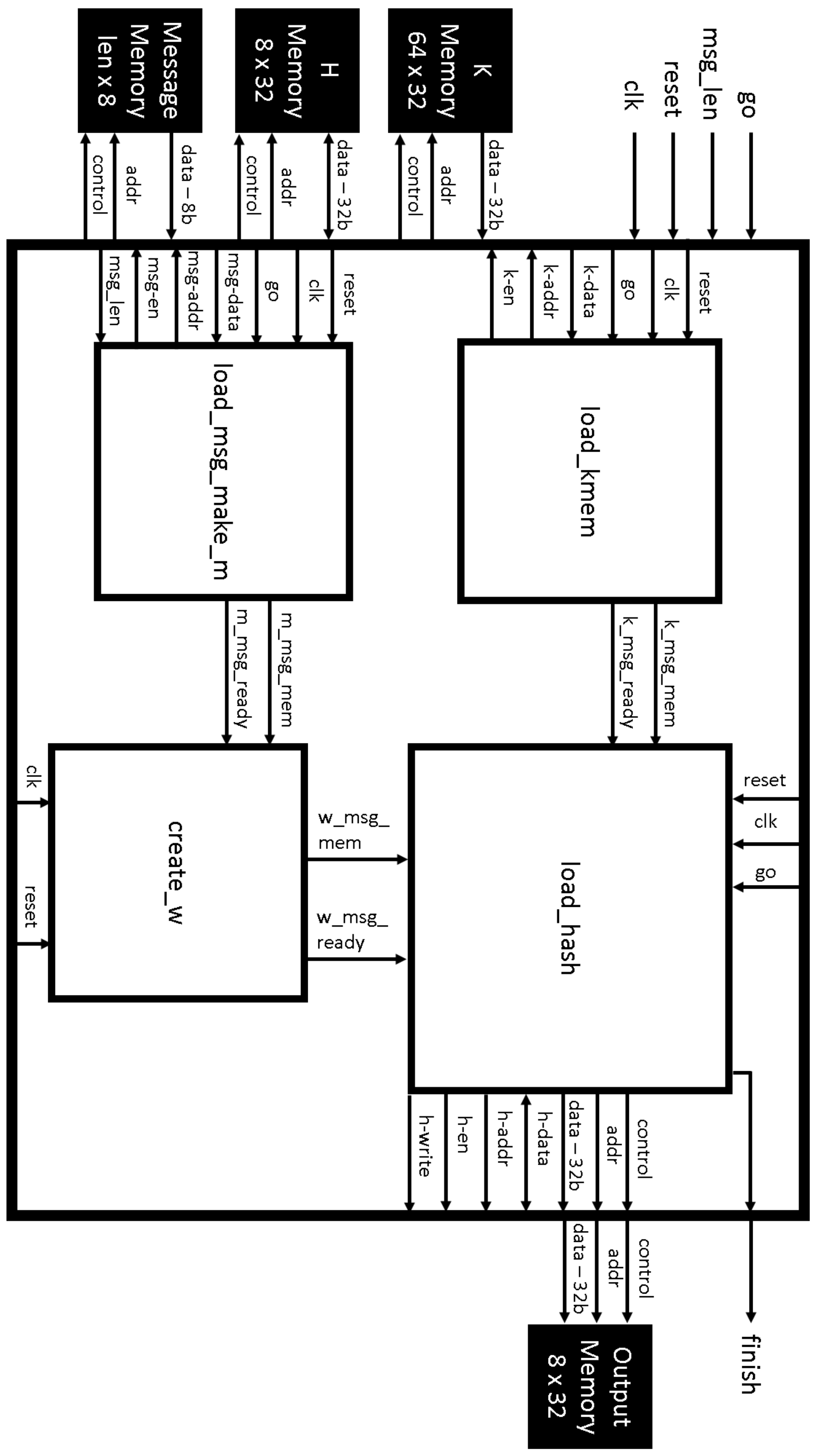
- The outer box in the block diagram is the top level module with other modules listed below instantiated within the top level module. Each module will have an internal FSM that will be controlling the sequence of operations to be performed.

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.


go
msg_len
reset
clk
K Memory 64 x 32
data – 32b
addr
control
H Memory 8 x 32
data – 32b
addr
control
Message Memory len x 8
data – 8b
addr
control
reset
clk
go
k-data
k-addr
k-en
load_kmem
k_msg_mem
k_msg_ready
reset
clk
go
msg-data
msg-addr
msg-en
msg_len
load_msg_make_m
m_msg_mem
m_msg_ready
load_hash
reset
clk
go
w_msg_ mem
w_msg_ ready
create_w
clk
reset
control
addr
data – 32b
h-data
h-addr
h-en
h-write
control
addr
data – 32b
Output Memory 8 x 32
finish