Project 2

RSA Encryption and Decryption module needs to be built. Example RSA encryption/decryption is:

given message M = 9

encryption: ( private key 3 and public key 33 )

C = 93 mod 33 = 3

decryption: ( private key 7 and public key 33 ) M = 37 mod 33 = 9

We need to do two operations regardless of Encryption or Decryption. One exponentiation operation and one mod operation. We already have the exponentiation operation from HW7 and modulus operation from HW8. You HAVE to use those modules from the HW7 and HW8 to build project 2. Incase you were not able to get HW7 or HW8 to work, then one will be provided. The Proj2 module needs implement an FSM, one instance of HW7 and one instance of HW8. Some details are provided later in the write up.

The high level module has 5 inputs Input private\_key = 16 bits

Input public\_key = 16 bits. Input message\_val = 16 bits. Input clk - 1 bit

Input Start - 1 bit Input Rst - 1 bit Output Cal\_done 1 bit Output Cal\_val 16 bit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* sample output that your testbench has to print out \*\*\*\*\*\*\*\*\*\*\*\*

Encoding….

Input private key = 3

Input Public key = 33

Input message = 9

Output value = 3

Decoding…..

Input private key = 7

Input Public key = 33

Input message = 3

Output value = 9

All I need to see is an FSM. The FSM caputures output in state 1 when start == 1. The FSM needs to generate the control +data signals for HW7 in state 2 and wait for the done signal in state 3. Then generate control singals for HW8 is state 4 and wait for done signal in state 5. In state 6 set cal\_val and cal\_done signals. This is the very basic that needs to be implemented. I have tried to come up with more details. But if this is too confusing then stick to the 6 states mentioned before and implement that in way it suits you.

At reset the FSM resets all the registers.

Capture\_state: When Start goes high, the Cal\_val and Cal\_done are set to zero. The two keys and the message needs to copied to internal registers. No need to check Start signal in the following states.

Exponent\_state1: feed the values to the myexp module from hw7 and raise the load signal. Exponenet\_state2: set the load signal to low

Exponent\_state3: monitor myexp done signal, if done is low jump back to state2 and keep bouncing between state 2 and 3

Add more/less states as needed.

Mod\_state1: feed the values to the mymodfunc module and raise the load signal. Mod\_state2: set the load signal to low

Mod\_state3: monitor myexpo done signal, if done is low jump back to state2 and keep bouncing between state 2 and 3

Add more states if you need

Cal\_done\_state1: set Cal\_done to high and the output appears on Cal\_val Cal\_done\_state2: set Cal\_done to high and the output appears on Cal\_val Cal\_done\_state3: set Cal\_done to high and the output appears on Cal\_val Go back to Capture\_state.

Make a test bench to show that the module does the encryption of the example shown at the start of the write up. In the testbench have an always statement that prints the state of the state machine. Also print the inputs and the final outputs.

Submit the project and tesbench doing the **encryption and decryption** as one file proj2.v.