Working directories:

/home/linux/ieng6/ee260c/aksuresh/project1 and /home/linux/ieng6/ee260c/skkrishn/AES_128bit/

Problem statement:

We outsourced a part of the AES co-processor encryption process to the hardware. Hence, some specified functions had to be coded on Verilog. This essentially is supposed to speed up the cryptographic computations in the MAC layer of the 802.11 Wireless LAN protocol.

In the 1st design, the area had to be minimized with a clock cycle time constraint of <u>20 ns.</u>

In the 2nd design, the area delay product had to be minimized with a clock cycle time constraint of <u>15ns.</u>

AES hardware description:

The AES uses the Rijnaldel algorithm which takes in a cipher of 128 bits in this design to encrypt data.

The 4 key transformations that are performed in hardware in this projects are:

1. Add round key (updated for every round with its key specific algorithm).
2. Add sub-bytes. The nibbles are provided as input to an “sbox” and a new value is subbed in its place according to the contents in the sbox.
3. Shift rows. The 2nd row is then right shifted once, the 3rd row twice, and the 4th row thrice.
4. The columns are then mixed after multiplication of each column with a constant. Galois field multiplication is used to accomplish this.

The data flow can be depicted as,

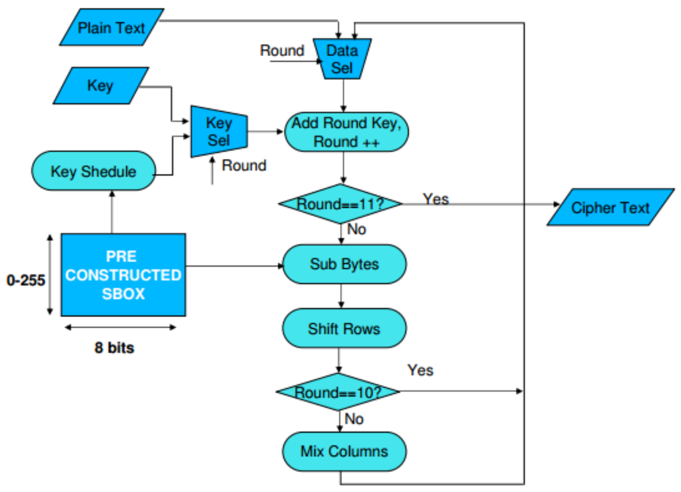


Fig. 1: The algorithm implemented on hardware

The principal aim of this project was to schedule operations appropriately into FSMs such that performance and cost optimizations were made with clock constraints.

Design flow:

Only the Co-processor code is written in this Verilog file. However, the interface to the AMBA bus and the DPSRAM still has to be done in the hardware outsourcing. The plain address, frame size, cipher keys and the cipher key address are provided as inputs along with the memory inputs which have the actual data stored according to the test-bench provided to us. The inputs are read by the hardware code after which the encryption begins. The outputs are written back on to the DPSRAM after encryption.

Working idea:

Part A: In part A, reducing the area was the only concern. Hence, the design was made to use ample number of clock cycles and a large period. While coding, each state had non-blocking assignments inside the states to reduce hardware and states (hardware) were reused to avoid generating hardware for similar operations.

A mix column function, and a Galois field multiplier were written as modules to be called upon whenever required. Hence, the hardware was reused.

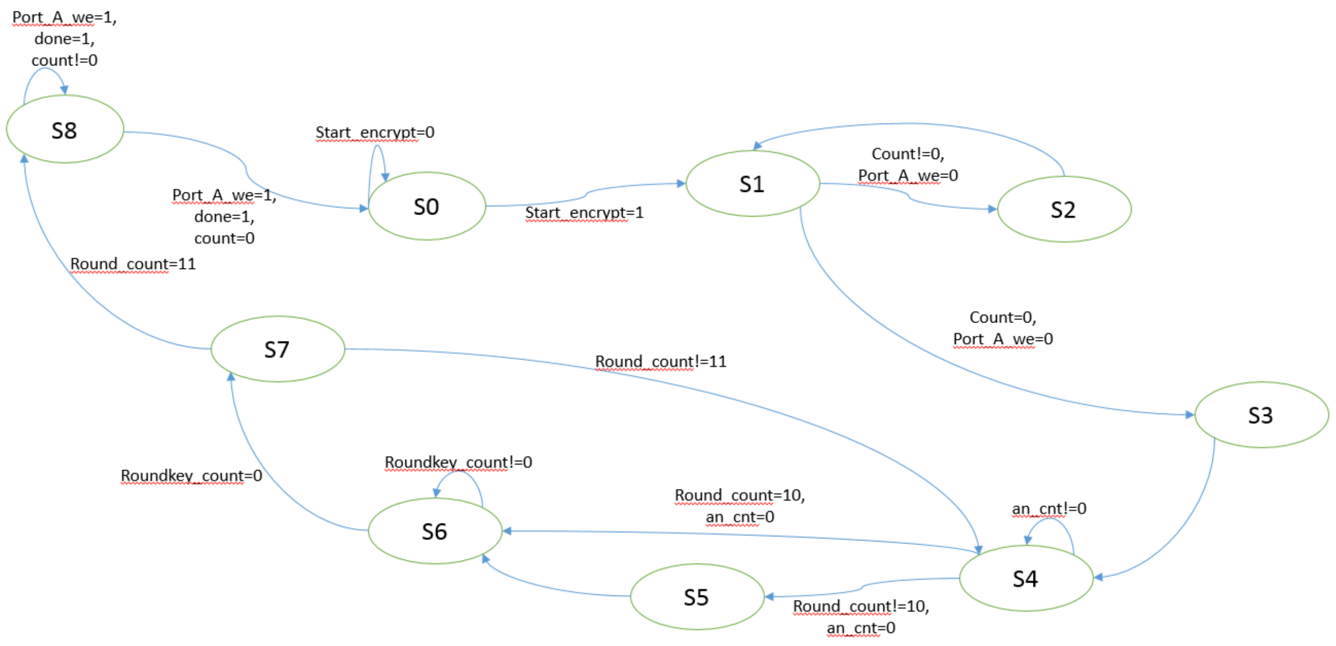
However, the sbox values did take up more hardware since it had to be hardcoded.

Part B: In part B, performance had to be factored in too which meant the clock period and the number of clock cycles had to be as low as possible.

Hence, hardware reuse was cut down by having more parallel computation and increasing the area to an extent.

State diagrams/optimizations:

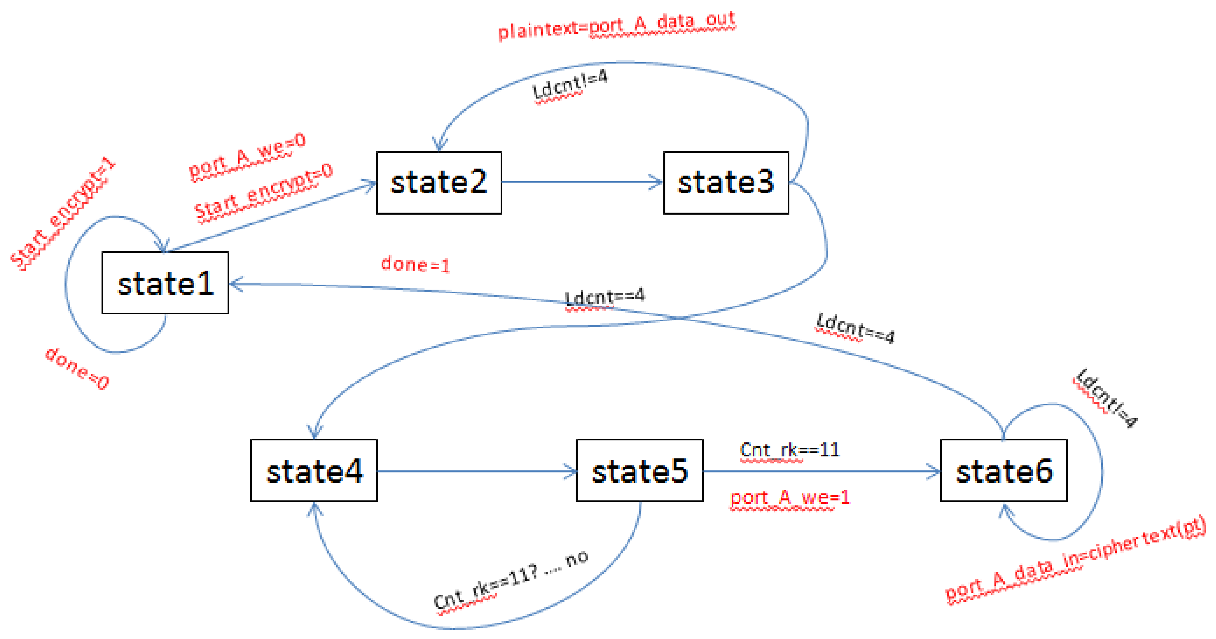
Part A:



Description:

- State0: This is the initial state. The encryption process, including the data read process starts when the start_encrypt is 1.
- State1: The data is read is this stage. The port address is incremented with a state diagram inside the state. A counter counts down to 0 to indicate that the data read process is complete.
- State2: A buffer state to give time for the data to be read.
- State3: After the data is read, the operations begin in state 3. The round key is added to the plain text data for the 1st round in this state. The state then shifts to state 4
- State4: Two operations i.e. the Sub-bytes and the Shift row operations are performed simultaneously by reading the sbox value and assigning it to the correct temporary register. Again, a counter counts down to 0 to cover all the values.
- State5: This is the mix_column step. Since the mix_column step is skipped for the last round of plain text encryption, if the round count is 10, it goes directly to the round key state. Otherwise, it computes the value using a function. It is called for every round and thus saves some hardware.
- State6: This state generates the round key for every round of encryption. The old key is replaced by the new key every round.
- State7: The round key is added to the text in this state. For the last round i.e. round 11, the values from state 4 is directly used to skip the mix_column state.
- State8: The data is written on to the DPSRAM in this state after incrementing the address by having a state machine and counter counting down the states. **The done bit is assigned to 1 for 1 clock cycle.**

__Part B__:



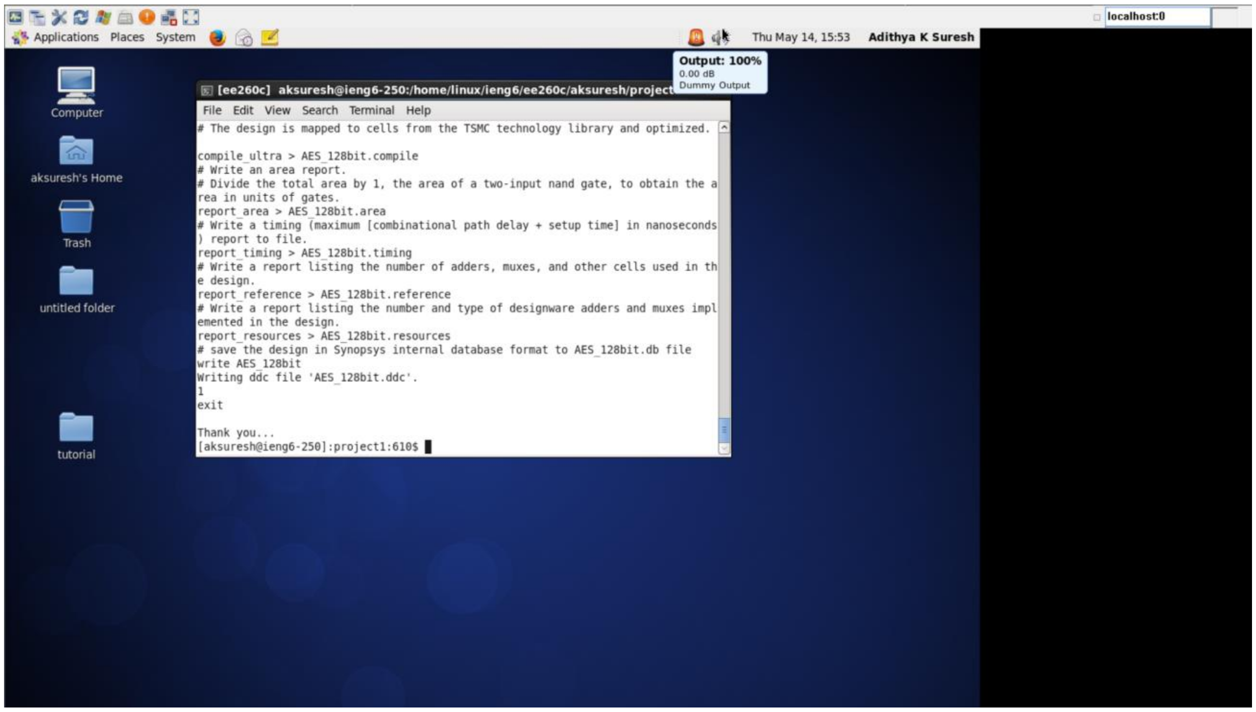
Description:

- State 1: In this state, we check whether the encryption process for that frame has begun. If so then we make the port_A_write output low so that we can read the words.
- State2: In this state, we send the plaintext address to the port_A_addr. We then wait for one cycle to get the data. For which we make a transition to state 3. We then read the data and decrement the counter ldcnt. This process repeats itself till all the words are read. In this stage we also read key0-3 and update our cipher key matrix.
- State 3: The data isn't available immediately. So that's why we need to make a transition to state three.
- State 4: Once we have both , the cipher key and the plain text, we then start the encryption process . In this stage we calculate the round key (if needed) and also proceed till the shift row stage of the plain text.
- Stage 5: in this stage we execute the mix column stage and add the round key to get the input for the next round. We loop through stage 4 and 5 till we have ccompleted the encryption process for that frame.
- Stage 6: In this stage we begin writing to the DRAM the port_A_we signal is made high to initiate the write process, we calculate the destination address and send the required data to the port while incrementing ldcnt each time. After writing the last word we make done bit 1.

For this part of the question since the delay is the most important factor out entire focus was to reduce the number of clock cycles to as low as possible and implement it in an efficient manner. Keeping in mind the tradeoff between Area and delay, we zeroed on one of the two resource allocations: 4 sboxes,16 galois fields multipliers and 36 XOR gates and 20 sboxes, 64 galois field multipliers and 144 XOR gates. We choose to go with the second implementation because we were able to execute the operation three times faster with a limited penalty on area. The optimizations in this stage involved taking advantage of the highly parallelized nature of the encryption algorithm to minimize number registers needed.

Model-sim traces and simulations:

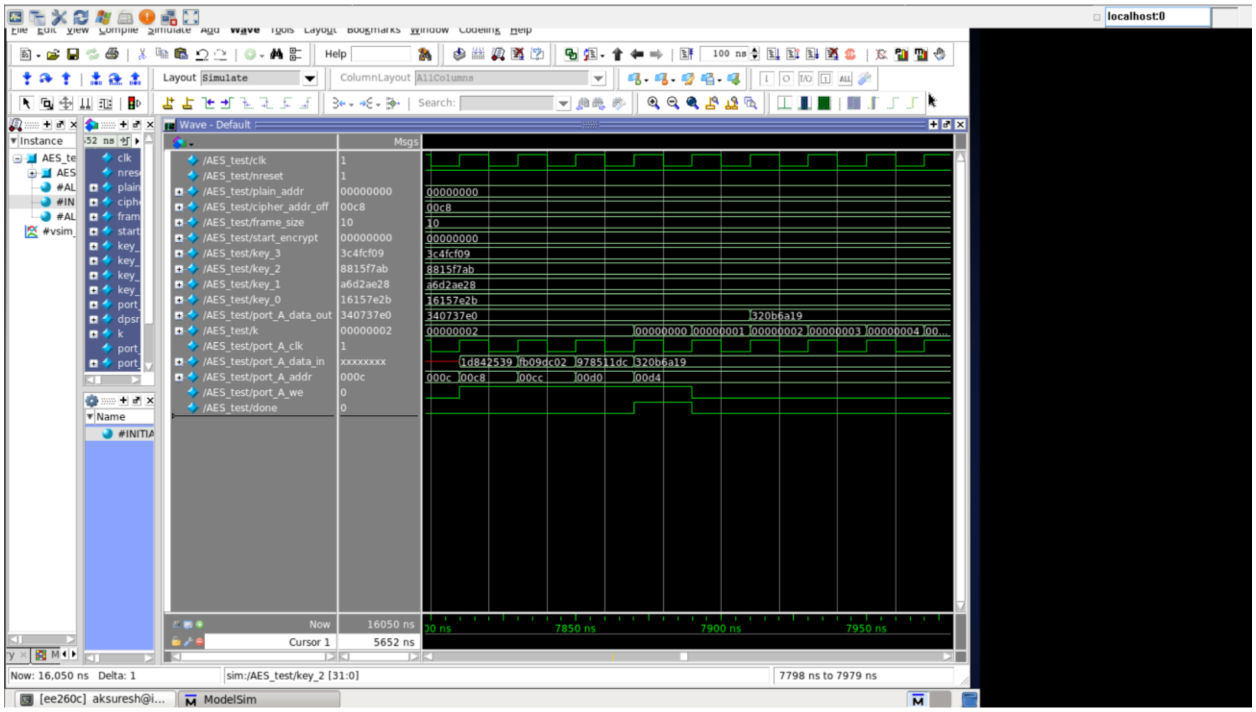
Part A:



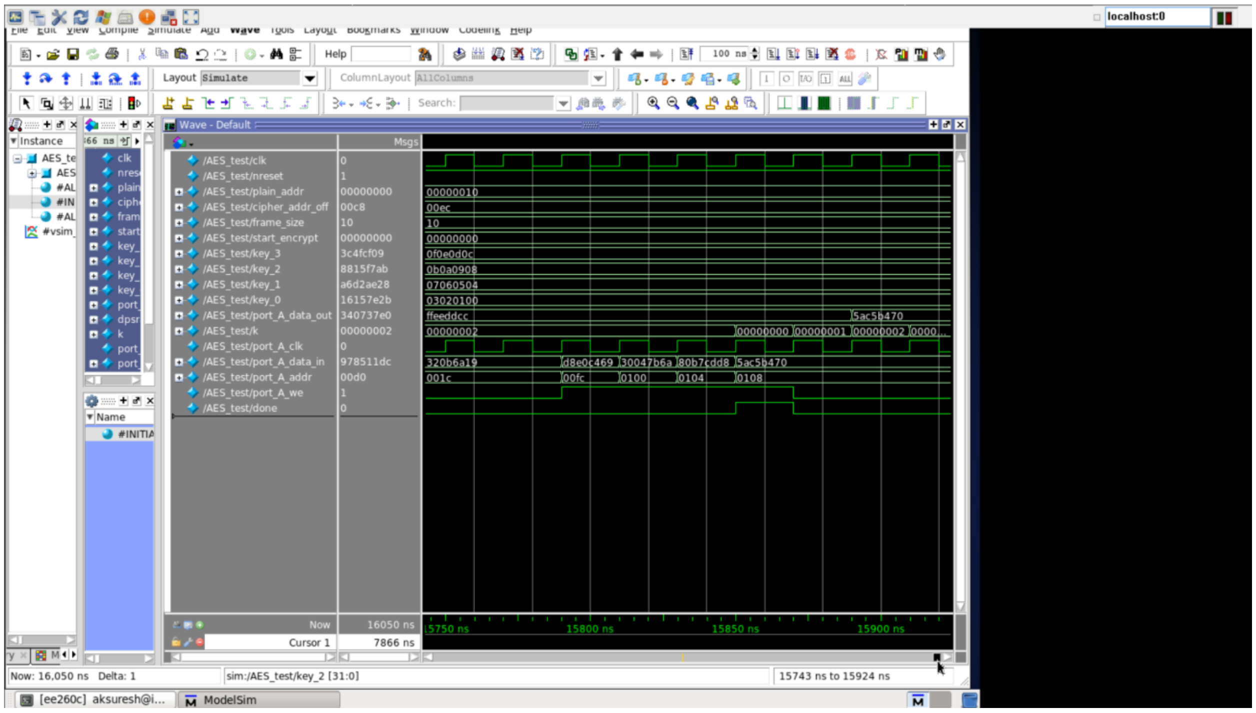
```
# The design is mapped to cells from the TSMC technology library and optimized.

compile_ultra > AES_128bit.compile
# Write an area report.
# Divide the total area by 1, the area of a two-input nand gate, to obtain the a
rea in units of gates.
report_area > AES_128bit.area
# Write a timing (maximum [combinational path delay + setup time] in nanoseconds
) report to file.
report_timing > AES_128bit.timing
# Write a report listing the number of adders, muxes, and other cells used in th
e design.
report_reference > AES_128bit.reference
# Write a report listing the number and type of designware adders and muxes impl
emented in the design.
report_resources > AES_128bit.resources
# save the design in Synopsys internal database format to AES_128bit.db file
write AES_128bit
Writing ddc file 'AES_128bit.ddc'.
1
exit

Thank you...
[aksuresh@ieng6-250]:project1:610$
```



```
/AES_test/clk                1
/AES_test/nreset             1
/AES_test/plain_addr         00000000
/AES_test/cipher_addr_off    00c8
/AES_test/frame_size         10
/AES_test/start_encrypt      00000000
/AES_test/key_3              3c4fcf09
/AES_test/key_2              8815f7ab
/AES_test/key_1              a6d2ae28
/AES_test/key_0              16157e2b
/AES_test/port_A_data_out    340737e0
/AES_test/k                  00000002
/AES_test/port_A_clk         1
/AES_test/port_A_data_in     xxxxxxxx
/AES_test/port_A_addr        000c
/AES_test/port_A_we          0
/AES_test/done               0
```

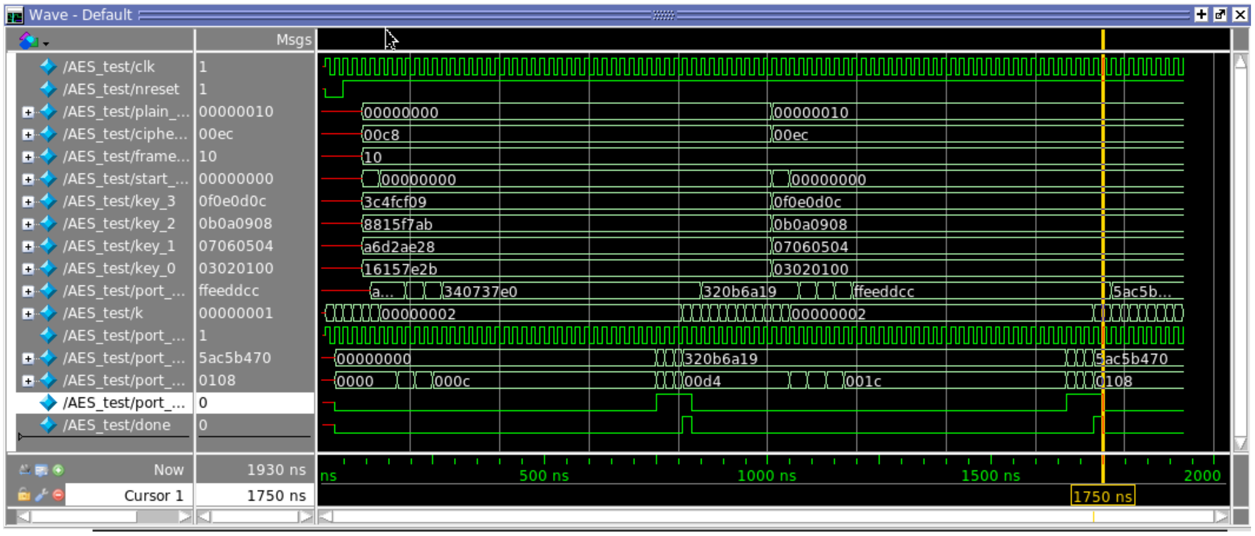


	Msgs
/AES_test/clk	0
/AES_test/nreset	1
/AES_test/plain_addr	00000000
/AES_test/cipher_addr_off	00c8
/AES_test/frame_size	10
/AES_test/start_encrypt	00000000
/AES_test/key_3	3c4fcf09
/AES_test/key_2	8815f7ab
/AES_test/key_1	a6d2ae28
/AES_test/key_0	16157e2b
/AES_test/port_A_data_out	340737e0
/AES_test/k	00000002
/AES_test/port_A_clk	0
/AES_test/port_A_data_in	978511dc
/AES_test/port_A_addr	00d0
/AES_test/port_A_we	1
/AES_test/done	0
Now	16050 ns
Cursor 1	7866 ns

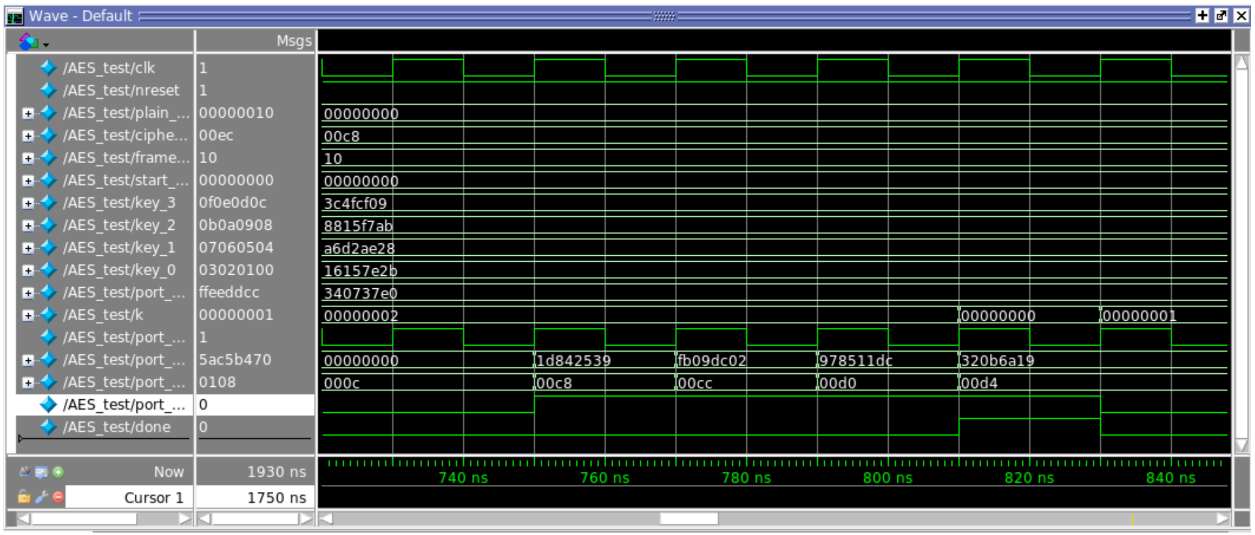
Now: 16,050 ns Delta: 1 sim:/AES_test/key_2 [31:0] 15743 ns to 15924 ns

It is seen that the simulations pass the test bench.

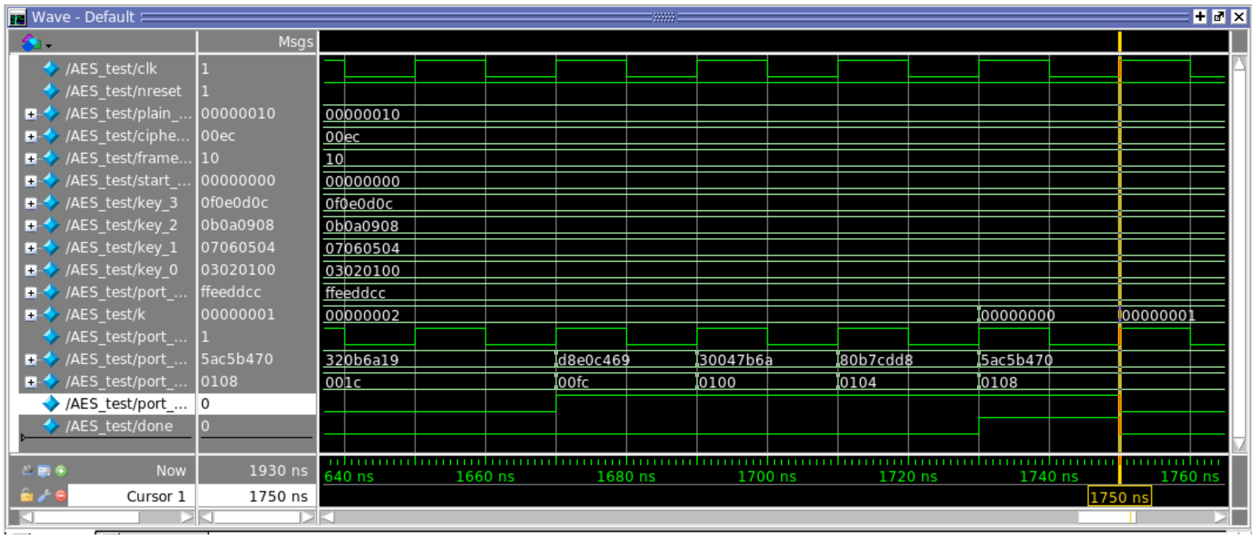
Part B:



	Msgs
/AES_test/clk	1
/AES_test/nreset	1
/AES_test/plain_...	00000010
/AES_test/ciphe...	00ec
/AES_test/frame...	10
/AES_test/start_...	00000000
/AES_test/key_3	0f0e0d0c
/AES_test/key_2	0b0a0908
/AES_test/key_1	07060504
/AES_test/key_0	03020100
/AES_test/port_...	ffeeddcc
/AES_test/k	00000001
/AES_test/port_...	1
/AES_test/port_...	5ac5b470
/AES_test/port_...	0108
/AES_test/port_...	0
/AES_test/done	0
Now	1930 ns
Cursor 1	1750 ns



```
Wave - Default
                      Msgs
/AES_test/clk         1
/AES_test/nreset      1
/AES_test/plain_...   00000010
/AES_test/ciphe...    00ec
/AES_test/frame...    10
/AES_test/start_...   00000000
/AES_test/key_3       0f0e0d0c
/AES_test/key_2       0b0a0908
/AES_test/key_1       07060504
/AES_test/key_0       03020100
/AES_test/port_...    ffeeddcc
/AES_test/k           00000001
/AES_test/port_...    1
/AES_test/port_...    5ac5b470
/AES_test/port_...    0108
/AES_test/port_...    0
/AES_test/done        0
Now                   1930 ns
Cursor 1              1750 ns
```



```
Wave - Default
                      Msgs
/AES_test/clk         1
/AES_test/nreset      1
/AES_test/plain_...   00000010
/AES_test/ciphe...    00ec
/AES_test/frame...    10
/AES_test/start_...   00000000
/AES_test/key_3       0f0e0d0c
/AES_test/key_2       0b0a0908
/AES_test/key_1       07060504
/AES_test/key_0       03020100
/AES_test/port_...    ffeeddcc
/AES_test/k           00000001
/AES_test/port_...    1
/AES_test/port_...    5ac5b470
/AES_test/port_...    0108
/AES_test/port_...    0
/AES_test/done        0
Now                   1930 ns
Cursor 1              1750 ns
```

***Results*:**

Part A: the **total cell area=7743 um^2.**

Part B: Area*Delay Calculation:

- Clock period of testbench=20ns.
- Time between the instance where nreset goes high and Instance second done bit goes high:1680ns
- **Number of clock cycles=1680/20=84 clock cycles.**
- The clock period comes from the timing report. Subtracting the slack from the test bench clock period, **clock period is 12.31ns**
- Delay=12.31x84=1034.34ns=1.03434us
- Area comes from the area report
- **Total Cell area = 15223.350um^2**

Area delay product= 15745.7578um^2-us