HDL Digital Design (Graduate Level)
Spring 2024

HOMEWORK
REPORT

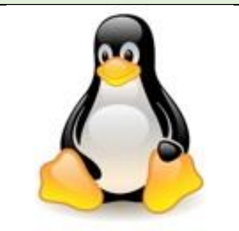
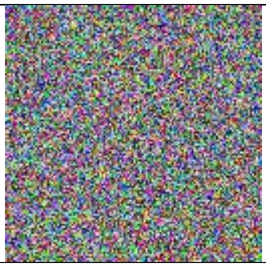
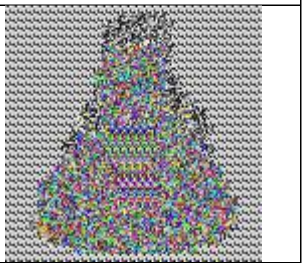
Must do self-checking before submission:

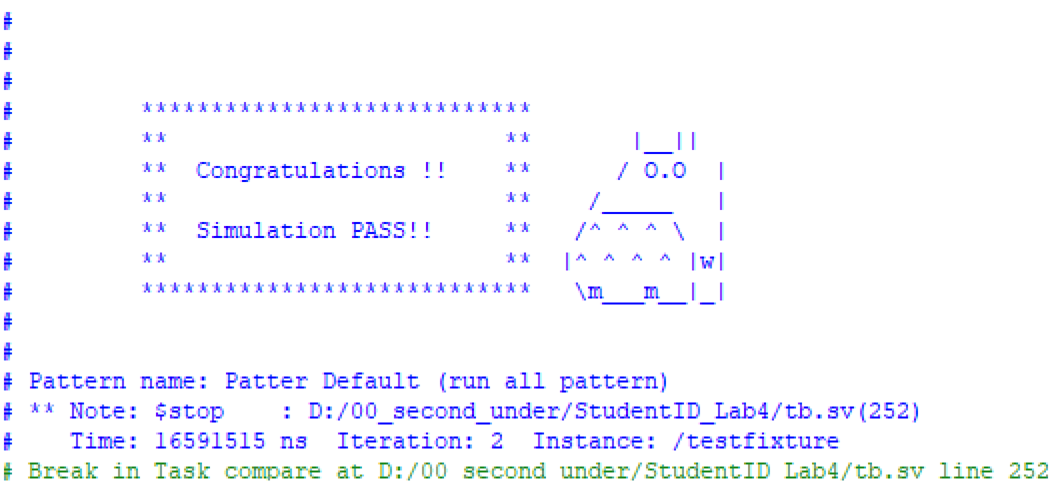
□ Compress all files described in the problem into one zip file.
□ All files can be compiled under ModelSim environment.
□ All port declarations comply with I/O port specifications.
□ Organize files according to File Hierarchy Requirement
□ No waveform files or project files in deliverables

Due Date: 2024/03/21 8:59 a.m.

Student name: 蔡承哲
Student ID: Q36111150

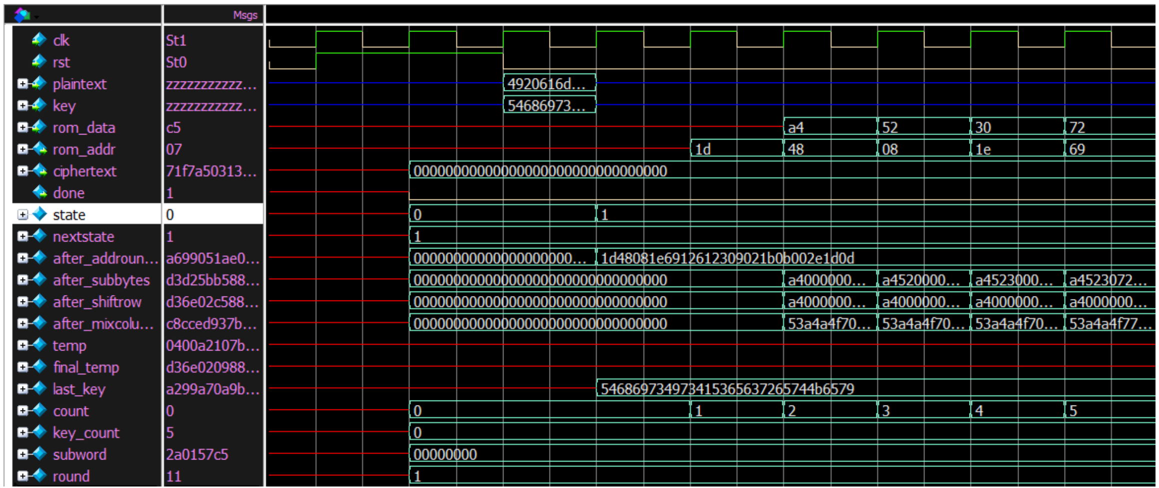
1. Paste simulation result on the terminal and result of two cipher image.

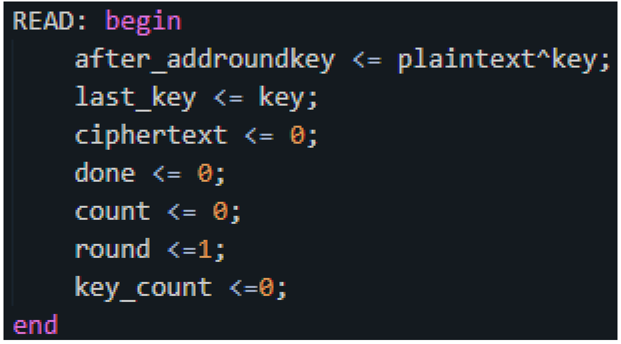
	mount	tux
Original		
Cipher		



```
#
#
#
#         ****************************
#         **                        **       |__||
#         **  Congratulations !!    **      / O.O  |
#         **                        **    /_____   |
#         **  Simulation PASS!!     **   /^ ^ ^ \  |
#         **                        **  |^ ^ ^ ^ |w|
#         ****************************   \m___m__|_|
#
#
# Pattern name: Patter Default (run all pattern)
# ** Note: $stop    : D:/00_second_under/StudentID_Lab4/tb.sv(252)
#    Time: 16591515 ns  Iteration: 2  Instance: /testfixture
# Break in Task compare at D:/00 second under/StudentID Lab4/tb.sv line 252
```

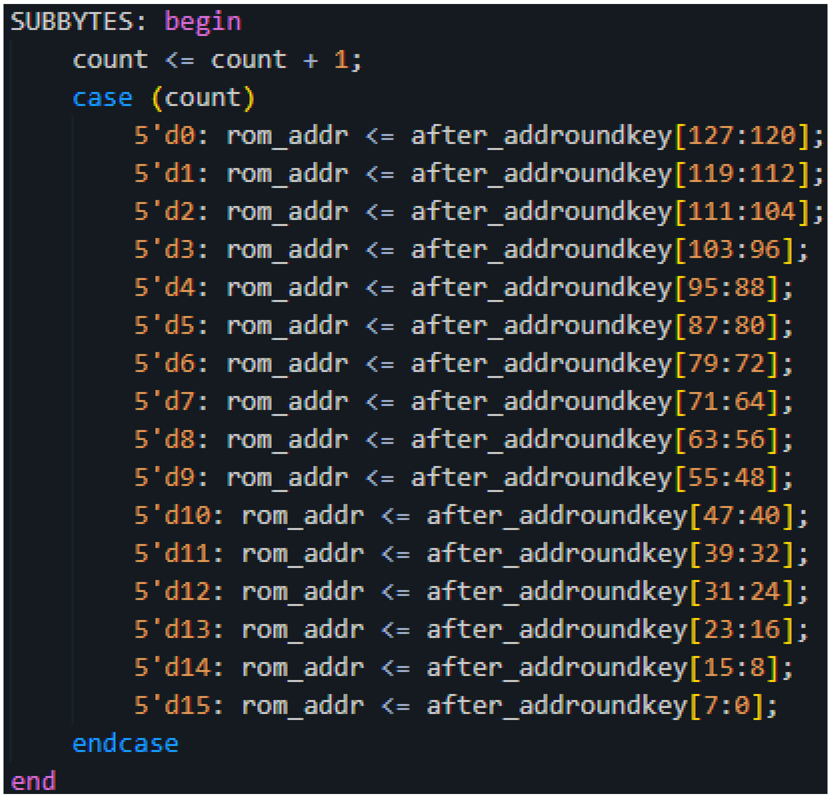
2. Explain the result by waveform.


Msgs
clk St1
rst St0
plaintext zzzzzzzzzzz... 4920616d...
key zzzzzzzzzzz... 54686973...
rom_data c5 a4 52 30 72
rom_addr 07 1d 48 08 1e 69
ciphertext 71f7a50313... 00000000000000000000000000000000
done 1
state 0 0 1
nextstate 1 1
after_addroun... a699051ae0... 0000000000000000000... 1d48081e6912612309021b0b002e1d0d
after_subbytes d3d25bb588... 00000000000000000000000000000000 a4000000... a4520000... a4523000... a4523072...
after_shiftrow d36e02c588... 00000000000000000000000000000000 a4000000... a4000000... a4000000... a4000000...
after_mixcolu... c8cced937b... 00000000000000000000000000000000 53a4a4f70... 53a4a4f70... 53a4a4f70... 53a4a4f77...
temp 0400a2107b...
final_temp d36e020988...
last_key a299a70a9b... 5468697349734153656372657 44b6579
count 0 0 1 2 3 4 5
key_count 5 0
subword 2a0157c5 00000000
round 11 1

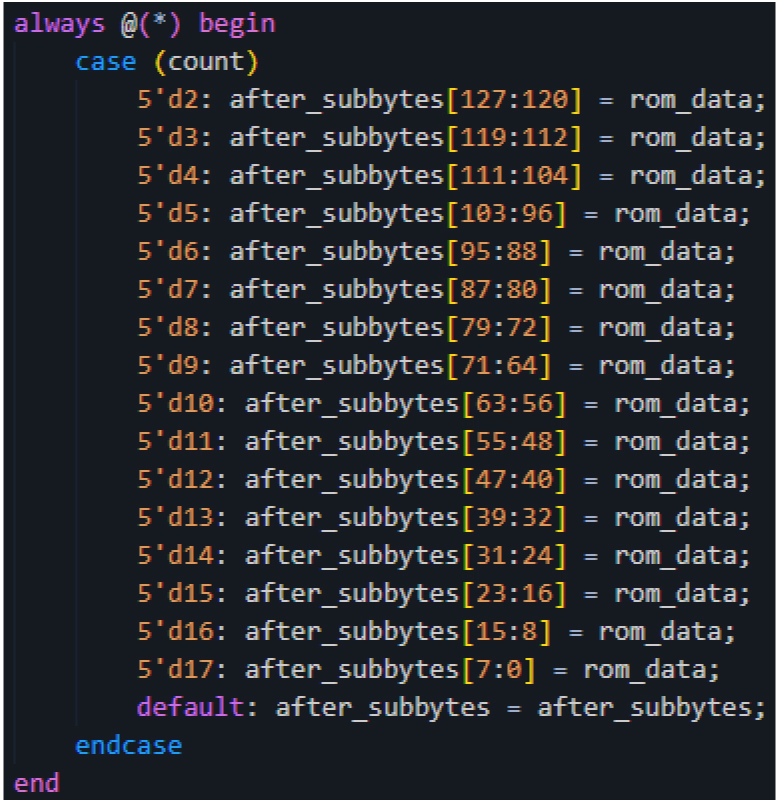


```
READ: begin
    after_addroundkey <= plaintext^key;
    last_key <= key;
    ciphertext <= 0;
    done <= 0;
    count <= 0;
    round <=1;
    key_count <=0;
end
```

首先在 state0(READ)的時候等待 plaintext 與 key 的輸入，然後把 plaintext^key 的結果賦值到 after_addroundkey。同時也把 key 賦值到 last_key，後續 update key 會用到。



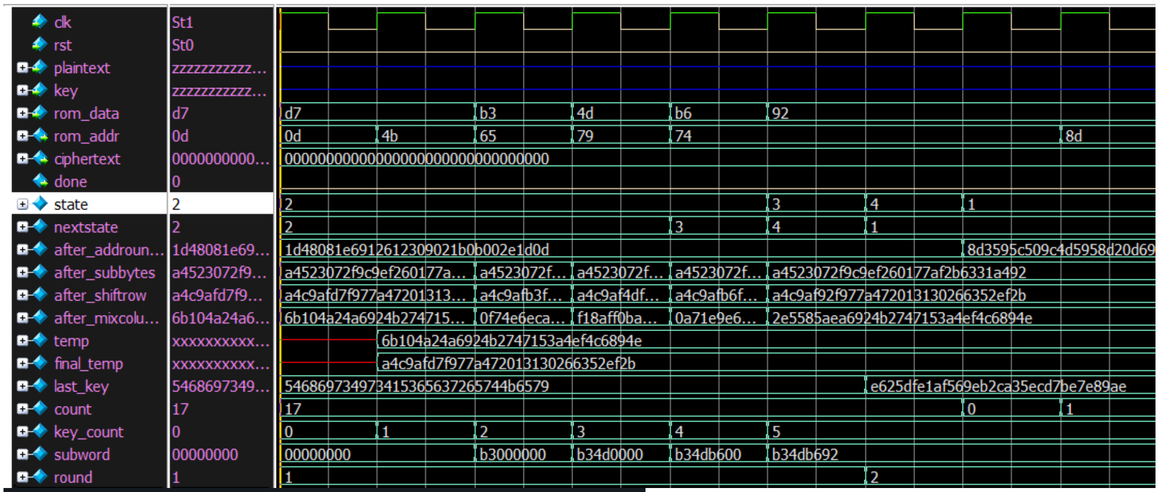
```
SUBBYTES: begin
    count <= count + 1;
    case (count)
        5'd0: rom_addr <= after_addroundkey[127:120];
        5'd1: rom_addr <= after_addroundkey[119:112];
        5'd2: rom_addr <= after_addroundkey[111:104];
        5'd3: rom_addr <= after_addroundkey[103:96];
        5'd4: rom_addr <= after_addroundkey[95:88];
        5'd5: rom_addr <= after_addroundkey[87:80];
        5'd6: rom_addr <= after_addroundkey[79:72];
        5'd7: rom_addr <= after_addroundkey[71:64];
        5'd8: rom_addr <= after_addroundkey[63:56];
        5'd9: rom_addr <= after_addroundkey[55:48];
        5'd10: rom_addr <= after_addroundkey[47:40];
        5'd11: rom_addr <= after_addroundkey[39:32];
        5'd12: rom_addr <= after_addroundkey[31:24];
        5'd13: rom_addr <= after_addroundkey[23:16];
        5'd14: rom_addr <= after_addroundkey[15:8];
        5'd15: rom_addr <= after_addroundkey[7:0];
    endcase
end
```



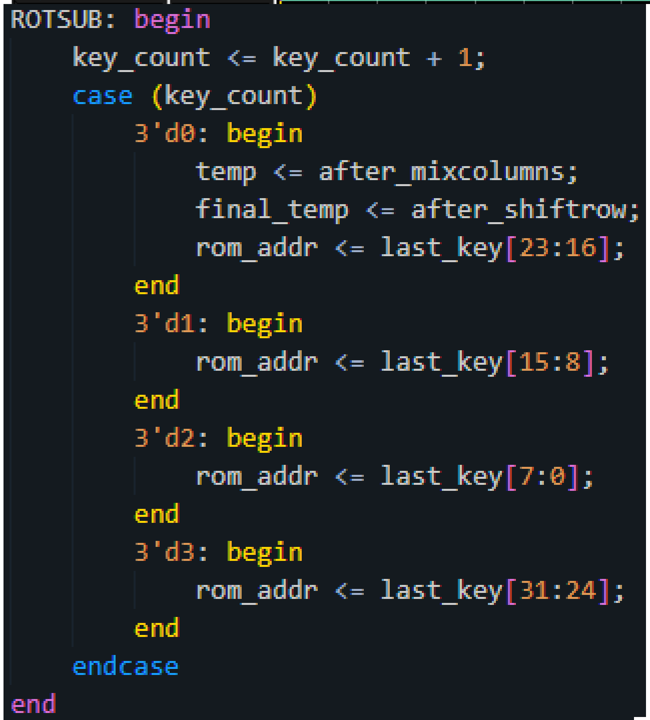
```
always @(*) begin
    case (count)
        5'd2: after_subbytes[127:120] = rom_data;
        5'd3: after_subbytes[119:112] = rom_data;
        5'd4: after_subbytes[111:104] = rom_data;
        5'd5: after_subbytes[103:96] = rom_data;
        5'd6: after_subbytes[95:88] = rom_data;
        5'd7: after_subbytes[87:80] = rom_data;
        5'd8: after_subbytes[79:72] = rom_data;
        5'd9: after_subbytes[71:64] = rom_data;
        5'd10: after_subbytes[63:56] = rom_data;
        5'd11: after_subbytes[55:48] = rom_data;
        5'd12: after_subbytes[47:40] = rom_data;
        5'd13: after_subbytes[39:32] = rom_data;
        5'd14: after_subbytes[31:24] = rom_data;
        5'd15: after_subbytes[23:16] = rom_data;
        5'd16: after_subbytes[15:8] = rom_data;
        5'd17: after_subbytes[7:0] = rom_data;
        default: after_subbytes = after_subbytes;
    endcase
end
```

接著就從 state0 跳到 state1(SUBBYTES)，這裡主要做的就是把 after_addroundkey 的每一個 byte 依序當

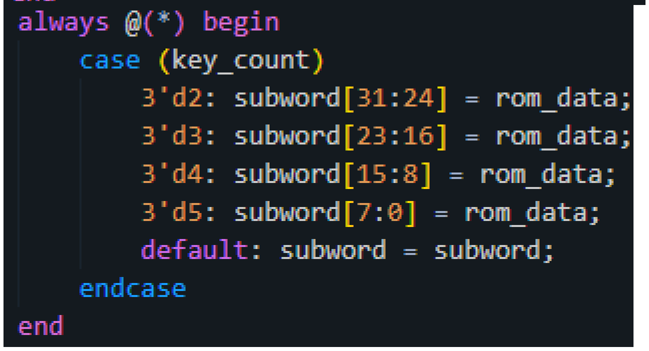
成 rom_addr 輸出，且等待 rom_data 回傳，依序存入 after_subbytes。



clk	St1
rst	St0
plaintext	zzzzzzzzzzz...
key	zzzzzzzzzzz...
rom_data	d7
rom_addr	0d
ciphertext	0000000000...
done	0
state	2
nextstate	2
after_addroun...	1d48081e69...
after_subbytes	a4523072f9...
after_shiftrow	a4c9afd7f9...
after_mixcolu...	6b104a24a6...
temp	xxxxxxxxxxx...
final_temp	xxxxxxxxxxx...
last_key	5468697349...
count	17
key_count	0
subword	00000000
round	1

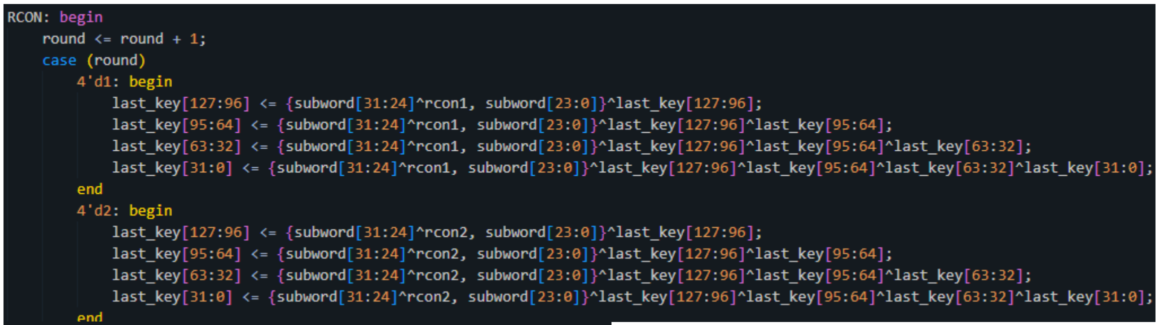


```
ROTSUB: begin
    key_count <= key_count + 1;
    case (key_count)
        3'd0: begin
            temp <= after_mixcolumns;
            final_temp <= after_shiftrow;
            rom_addr <= last_key[23:16];
        end
        3'd1: begin
            rom_addr <= last_key[15:8];
        end
        3'd2: begin
            rom_addr <= last_key[7:0];
        end
        3'd3: begin
            rom_addr <= last_key[31:24];
        end
    endcase
end
```

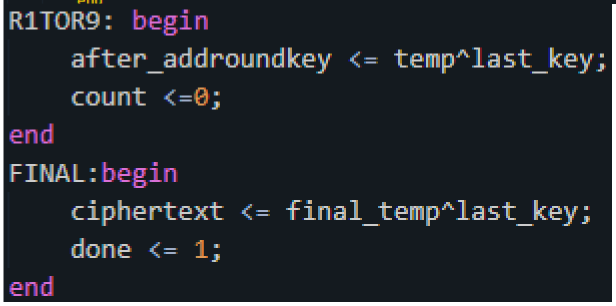


```
always @(*) begin
    case (key_count)
        3'd2: subword[31:24] = rom_data;
        3'd3: subword[23:16] = rom_data;
        3'd4: subword[15:8] = rom_data;
        3'd5: subword[7:0] = rom_data;
        default: subword = subword;
    endcase
end
```

因為做完 subbytes 後，我們還缺少新的 key 去做下一輪，所以 state1 跳到 state2(ROTSUB)，因為做完 subbytes 的同時，就會得到 after_mixcolumns，所以在這個 state 也會同時暫存 after_mixcolumns(前 9 輪會用到)與 after_shiftrow(最後一輪用)。經過這個 state 後就可以得到 subword。



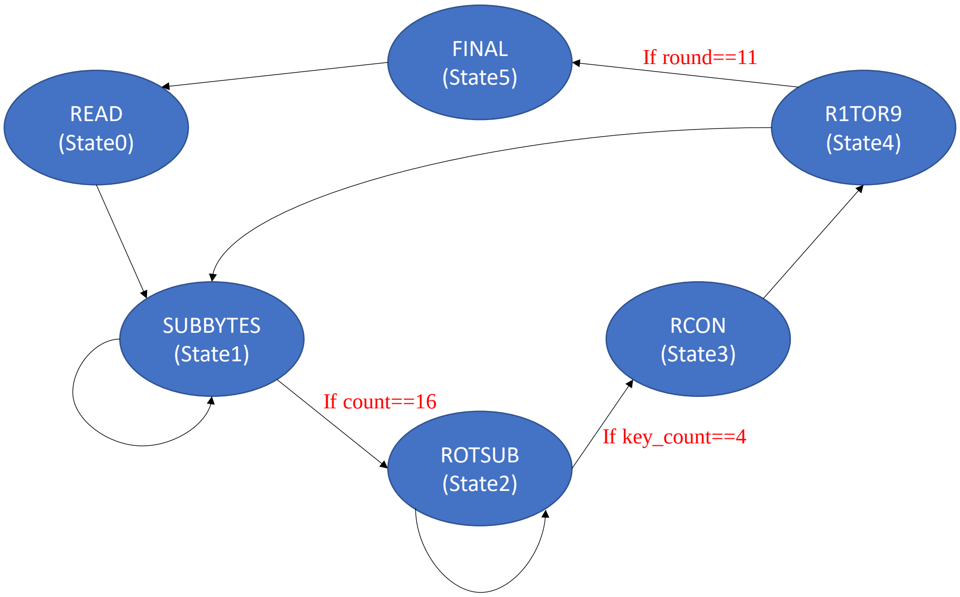
```
RCON: begin
    round <= round + 1;
    case (round)
        4'd1: begin
            last_key[127:96] <= {subword[31:24]^rcon1, subword[23:0]}^last_key[127:96];
            last_key[95:64] <= {subword[31:24]^rcon1, subword[23:0]}^last_key[127:96]^last_key[95:64];
            last_key[63:32] <= {subword[31:24]^rcon1, subword[23:0]}^last_key[127:96]^last_key[95:64]^last_key[63:32];
            last_key[31:0] <= {subword[31:24]^rcon1, subword[23:0]}^last_key[127:96]^last_key[95:64]^last_key[63:32]^last_key[31:0];
        end
        4'd2: begin
            last_key[127:96] <= {subword[31:24]^rcon2, subword[23:0]}^last_key[127:96];
            last_key[95:64] <= {subword[31:24]^rcon2, subword[23:0]}^last_key[127:96]^last_key[95:64];
            last_key[63:32] <= {subword[31:24]^rcon2, subword[23:0]}^last_key[127:96]^last_key[95:64]^last_key[63:32];
            last_key[31:0] <= {subword[31:24]^rcon2, subword[23:0]}^last_key[127:96]^last_key[95:64]^last_key[63:32]^last_key[31:0];
        end
```



```
R1TOR9: begin
    after_addroundkey <= temp^last_key;
    count <=0;
end
FINAL:begin
    ciphertext <= final_temp^last_key;
    done <= 1;
end
```

接著 state2 跳到 state3(RCON)產生新的 key，然後就再跳到 state4(R1TOR9)，把 temp(after_mixcolumns)^last_key 的結果再度傳入 after_addroundkey。之後就依序做每一輪，最後一輪，狀態跳到 state5，因為不需要做 mixcolumns，所以做後結果就是 final_temp(after_shiftrow)^last_key。

3. Draw your own Finite State Machine.



4. At last, please write the lesson learned from Lab4 and discuss why Cipher_tux still has contour on the image.
   - 學會使用狀態機
   - 因為這次的加密模式是使用 ECB mode，所以也是造成影像中有輪廓的原因。在影像的上下文中，因為具有相同顏色或陰影的影像區域將產生相同的加密區塊。