Brief Report : Serpent implementation

電機二 劉紀霆

一、架構

- Decryption
  - dec_round.v
  - decrypt.v
  - Inv_Sbox.v
- Encryption
  - round.v
  - Serpent.v
  - table.v
- tb.v
- report.pdf

TEST:

iverilog -o out .\tb.v

vvp out

二、結果

將 dir 設為 1，加密的結果如下(擷取前十個結果)，並與 serpent online cipher 比較如下圖

```
C:\Users\user\Desktop\NTU\108-summer\Cryptography\FPGA test\Serpent>vvp out
VCD info: dumpfile top.vcd opened for output.
6aef70fc137b2bb40ccbcddc0b9cc71d
c7358b9f22020ff974c7ec488fa143fd
ff30cc12fbf44f1ec52784b3c439a8f7
e5ae3a7ceb449f8cdcdf1133b56a936e
ef0304cb03b23593bc5f97319b9af606
2f0459b824d85b312afaed864ce4fea9
24cf71cd67e8d18f566d55552e95e064
2713364b9cf6f67abb790090839d0c8b
40a4dd066b9d8544c6958fc1d129c103
aac013ed43faa4b793573b326f09e600
```

00000000	1d	c7	9c	0b	dc	cd	cb	0c	b4	2b	7b	13	fc	70	ef	6a
00000010	fd	43	a1	8f	48	ec	c7	74	f9	0f	02	22	9f	8b	35	c7
00000020	f7	a8	39	c4	b3	84	27	c5	1e	4f	f4	fb	12	cc	30	ff
00000030	6e	93	6a	b5	33	11	df	dc	8c	9f	44	eb	7c	3a	ae	e5
00000040	06	f6	9a	9b	31	97	5f	bc	93	35	b2	03	cb	04	03	ef
00000050	a9	fe	e4	4c	86	ed	fa	2a	31	5b	d8	24	b8	59	04	2f
00000060	64	e0	95	2e	55	55	6d	56	8f	d1	e8	67	cd	71	cf	24
00000070	8b	0c	9d	83	90	00	79	bb	7a	f6	f6	9c	4b	36	13	27
00000080	03	c1	29	d1	c1	8f	95	c6	44	85	9d	6b	06	dd	a4	40
00000090	00	e6	09	6f	32	3b	57	93	b7	a4	fa	43	ed	13	c0	aa

二者結果相同。（注意下圖要由右向左兩位兩位讀）

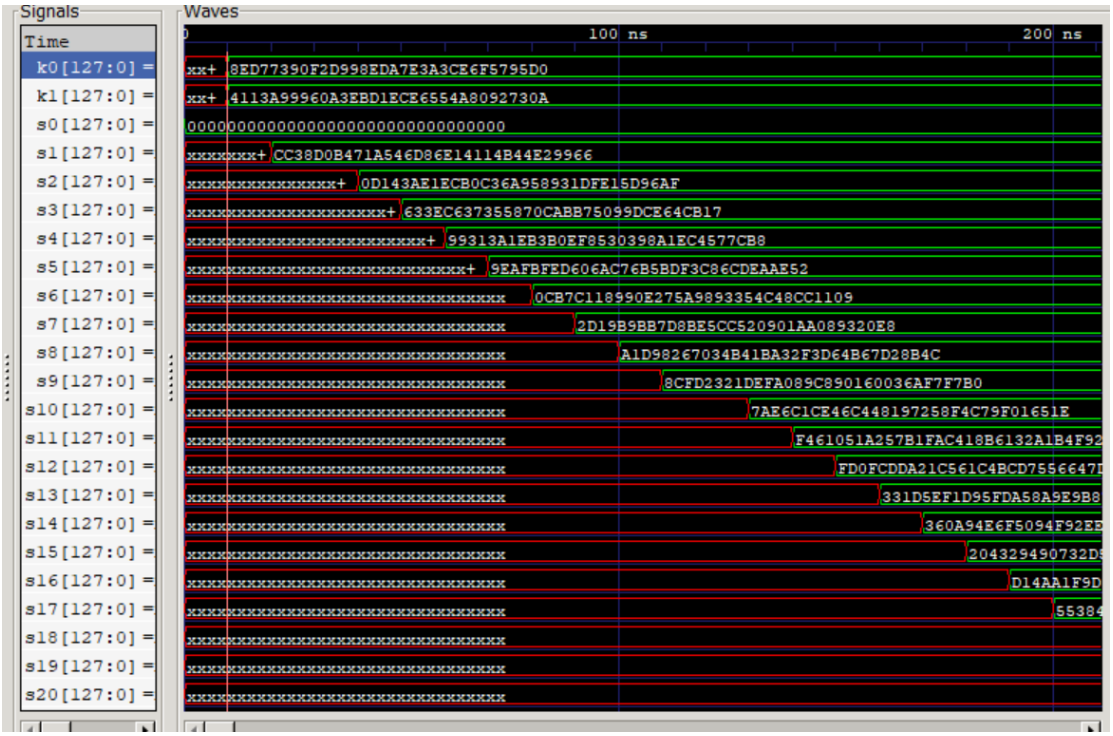
將 dir 設為 0，解密的結果如下(擷取前十個結果)，並與 serpent online cipher 比較如下圖

```
C:\Users\user\Desktop\NTU\108-summer\Cryptography\FPGA test\Serpent>vvp out
VCD info: dumpfile top.vcd opened for output.
45624f975efd32dcfe7689e4c7e984fe
3d3a4c8e8ea61a689d9b253ddc00023c
80fae975129786fd773af741564a1a89
4e157ca48a28aee746c02fb9d68e0734
25a0114e5056ac542c51e11bf7d8505e
684f276f3d1995bb5ed8ecf41c4744dc
6243163ab099455a3015d91937663871
bd39e611a6eff329da5469e6798b5a20
b87448cae6742c93559a5b63b51026d3
cfbd6388e625ad0f6ab149dd66eaaafc
```

00000000	fe	84	e9	c7	e4	89	76	fe	dc	32	fd	5e	97	4f	62	45
00000010	3c	02	00	dc	3d	25	9b	9d	68	1a	a6	8e	8e	4c	3a	3d
00000020	89	1a	4a	56	41	f7	3a	77	fd	86	97	12	75	e9	fa	80
00000030	34	07	8e	d6	b9	2f	c0	46	e7	ae	28	8a	a4	7c	15	4e
00000040	5e	50	d8	f7	1b	e1	51	2c	54	ac	56	50	4e	11	a0	25
00000050	dc	44	47	1c	f4	ec	d8	5e	bb	95	19	3d	6f	27	4f	68
00000060	71	38	66	37	19	d9	15	30	5a	45	99	b0	3a	16	43	62
00000070	20	5a	8b	79	e6	69	54	da	29	f3	ef	a6	11	e6	39	bd
00000080	d3	26	10	b5	63	5b	9a	55	93	2c	74	e6	ca	48	74	b8
00000090	fc	aa	ea	66	dd	49	b1	6a	0f	ad	25	e6	88	63	bd	cf

二者結果相同。

Gtkwave 圖，以 decryption 為例，encryption 同理



先存放好所有 prekey，則 Key schedule 會在同一個 cycle 完成，值得注意的是每個 round 的加密結果(s_r)都比前一 round 加密結果（$s_{(r-1)}$）都落後一個

clock，也就是 10 ns，這是因為每一個 Round 需要一個 Clock 做計算的緣故。

Verilog 的各 module 為同時進行運算，因此，只要 key 來的及算完，data 變化的速度最快可以為一個 clock cycle。

三、作法簡要說明

1. establish 32bits to 32bits Sbox

```
module S2_32 (clk, in, out);
    input clk;
    input [127:0] in;
    output [127:0] out;

    S2
        S0_0  (clk, in[127:124], out[127:124]),
        S0_1  (clk, in[123:120], out[123:120]),
        S0_2  (clk, in[119:116], out[119:116]),
        S0_3  (clk, in[115:112], out[115:112]),
```

2. Use wire and array to store prekey

```
assign pk[0] = ((w_n8 ^ w_n5 ^ w_n3 ^ w_n1 ^ phi ^ 0) << 11)|((w_n8 ^ w_n5 ^
assign pk[1] = ((w_n7 ^ w_n4 ^ w_n2 ^ pk[0] ^ phi ^ 1) << 11)|((w_n7 ^ w_n4 ^
assign pk[2] = ((w_n6 ^ w_n3 ^ w_n1 ^ pk[1] ^ phi ^ 2) << 11)|((w_n6 ^ w_n3 ^
assign pk[3] = ((w_n5 ^ w_n2 ^ pk[0] ^ pk[2] ^ phi ^ 3) << 11)|((w_n5 ^ w_n2
assign pk[4] = ((w_n4 ^ w_n1 ^ pk[1] ^ pk[3] ^ phi ^ 4) << 11)|((w_n4 ^ w_n1
assign pk[5] = ((w_n3 ^ pk[0] ^ pk[2] ^ pk[4] ^ phi ^ 5) << 11)|((w_n3 ^ pk[0
assign pk[6] = ((w_n2 ^ pk[1] ^ pk[3] ^ pk[5] ^ phi ^ 6) << 11)|((w_n2 ^ pk[1
assign pk[7] = ((w_n1 ^ pk[2] ^ pk[4] ^ pk[6] ^ phi ^ 7) << 11)|((w_n1 ^ pk[2
```

3. Use Sbox derive keys

```
S0_32
    a3 (i_clk,{pk[15] , pk[14] , pk[13] , pk[12]} , k3 ),
    a11(i_clk,{pk[47] , pk[46] , pk[45] , pk[44]} , k11),
    a19(i_clk,{pk[79] , pk[78] , pk[77] , pk[76]} , k19),
    a27(i_clk,{pk[111], pk[110], pk[109], pk[108]}, k27);
```

4. Use module IP to perform Initial Permutation

```
module IP(in, out);
    input [127:0] in;
    output [127:0] out;
    genvar i;
    generate
        for (i=0; i<32; i=i+1)
        begin
            assign out[4*i+0] = in[0+i];
            assign out[4*i+1] = in[32+i];
            assign out[4*i+2] = in[64+i];
            assign out[4*i+3] = in[96+i];
        end
    endgenerate
endmodule
```

5. define enc_round_0~7 to calculate each round (use key 0 ~ 30)

```
module enc_round_1(input [127:0] state_in,
                   output reg[127:0] state_out,
                   input [127:0] round_key,
                   input clk);
    wire[127:0] v1,v2;
```

6. final_round: with final permutation (use key 31,32)

```
module final_round(input [127:0] state_in,
                   output [127:0] state_out,
                   input [127:0] round_key1,
                   input [127:0] round_key2,
                   input clk);
```

7. package into one module !

```
enc_round_0
    e0(qq,s1,K0,i_clk), e8(s8,s9,K8,i_clk), e16(s16,s17,K16,i_clk), e24(s24,s25,K24,i_clk);
enc_round_1
    e1(s1,s2,K1,i_clk), e9(s9,s10,K9,i_clk), e17(s17,s18,K17,i_clk), e25(s25,s26,K25,i_clk);
enc_round_2
    e2(s2,s3,K2,i_clk), e10(s10,s11,K10,i_clk), e18(s18,s19,K18,i_clk), e26(s26,s27,K26,i_clk);
enc_round_3
    e3(s3,s4,K3,i_clk), e11(s11,s12,K11,i_clk), e19(s19,s20,K19,i_clk), e27(s27,s28,K27,i_clk);
enc_round_4
    e4(s4,s5,K4,i_clk), e12(s12,s13,K12,i_clk), e20(s20,s21,K20,i_clk), e28(s28,s29,K28,i_clk);
enc_round_5
    e5(s5,s6,K5,i_clk), e13(s13,s14,K13,i_clk), e21(s21,s22,K21,i_clk), e29(s29,s30,K29,i_clk);
enc_round_6
    e6(s6,s7,K6,i_clk), e14(s14,s15,K14,i_clk), e22(s22,s23,K22,i_clk), e30(s30,s31,K30,i_clk);
enc_round_7
```

8. decrypt: very similar

(make inverse S box)

```
module inv_S2_32 (clk, in, out);
    input clk;
    input [127:0] in;
    output [127:0] out;

    wire [127:0] ped_input; // after permutation
    wire [127:0] ped_output;
```

(dealing with each round)

```
module dec_final_round(input [127:0] state_in,
                       output [127:0] state_out,
                       input [127:0] round_key,
                       input clk);

    wire[127:0] v2,v3;

    inv_S7_32
        tmp (clk,state_in,v2);

    assign v3 = v2 ^ round_key;
    assign state_out = v3;
endmodule
```

(should be care of inverse linear transform, quiet different)

```
// inverse linear transform
assign x0 = {state_in[4:0], state_in[31:5]};
assign x1 = state_in[63:32];
assign x2 = {state_in[85:64], state_in[95:86]};
assign x3 = state_in[127:96];
assign  x_21 = x2 ^ x3 ^ (x1 << 7);
assign  x_01 = x0 ^ x1 ^ x3;
assign  x_31 = {x3[6:0], x3[31:7]};
assign  x_11 = {x1[0], x1[31:1]};
assign  x_32 = x_31 ^ x_21 ^ (x_01 << 3);
assign  x_12 = x_11 ^ x_01 ^ x_21;
assign  x_22 = {x_21[2:0], x_21[31:3]};
assign  x_02 = {x_01[12:0], x_01[31:13]};
```

9. package all of them

```
key_schedule
    KK (i_clk, i_key, k0, k1, k2, k3, k4, k5, k6,
        k7, k8, k9, k10, k11, k12, k13, k14, k15,
        k16, k17, k18, k19, k20, k21, k22, k23, k24,
        k25, k26, k27, k28, k29, k30, k31, k32);
encrypt
    ee(i_clk, i_data, k0, k1, k2, k3, k4, k5, k6,
     k7, k8, k9, k10, k11, k12, k13, k14, k15,
     k16, k17, k18, k19, k20, k21, k22, k23, k24,
     k25, k26, k27, k28, k29, k30, k31, k32, cipher);

decrypt
    dd(i_clk, i_data, k0, k1, k2, k3, k4, k5, k6,
      k7, k8, k9, k10, k11, k12, k13, k14, k15,
      k16, k17, k18, k19, k20, k21, k22, k23, k24,
      k25, k26, k27, k28, k29, k30, k31, k32, plain);
```

[附錄] 心得

這次的作業整體架構其實滿早就寫完了，但是一直對不到正確的加密結果，終於在最後一天找到了卡了五六個小時的 bug： 我把 linear transformation 的<<看成<<<，所以怎麼做怎麼錯....

這個暑假第一次學 verilog，很感謝學長用心的教學！並且在完成 Serpent 不斷 try and error 跟爬問題的過程之中，也對 Verilog 更熟悉了一些，收穫很多，謝謝學長！