

```
`include "timescale.v"
```

```
module sbox(clk,reset,data_i,decrypt_i,data_o);
```

```
input clk;
```

```
input reset;
```

```
input [7:0] data_i;
```

```
input decrypt_i;
```

```
output [7:0] data_o;
```

```
reg [7:0] data_o;
```

```
reg [7:0] inva;
```

```
reg [3:0] ah;
```

```
reg [3:0] al;
```

```
reg [3:0] ah2;
```

```
reg [3:0] al2;
```

```
reg [3:0] alxh;
```

```
reg [3:0] alph;
```

```
reg [3:0] d;
```

```
reg [3:0] ahp;
```

```
reg [3:0] alp;
```

```
reg [3:0] to_invert;
```

```
reg [3:0] next_to_invert;
```

```
reg [3:0] ah_reg;
```

```
reg [3:0] next_ah_reg;
```

```
reg [3:0] next_alph;
```

```
//registers:
```

```
always @(posedge clk or negedge reset)
```

```
begin
```

```
if(!reset)
```

```
begin
```

```
to_invert = (0);
```

```
ah_reg = (0);
```

```
alph = (0);
```

```
end
```

```
else
```

```
begin
```

```
to_invert = (next_to_invert);
```

```
ah_reg = (next_ah_reg);
```

```
alph = (next_alph);
```

```
end
```

```
end
```

```
//first_mux:
```

```
reg[7:0] first_mux_data_var;
```

```
reg[7:0] first_mux_InvInput;
```

```
reg[3:0] first_mux_ah_t,first_mux_al_t;
```

```
reg first_mux_aA,first_mux_aB,first_mux_aC,first_mux_aD;
```

```
always @(data_i or decrypt_i)
```

```
begin
```

```
    first_mux_data_var=data_i;
```

```
    first_mux_InvInput=first_mux_data_var;
```

```
    case(decrypt_i)
```

```
        1:
```

```
        begin
```

```
            //Apply inverse affine trasformation
```

```
            first_mux_aA=first_mux_data_var[0]^first_mux_data_var[5];first_mux_aB=first_mux_data_var[1]^first_mux_data_var[4];
```

```
            first_mux_aC=first_mux_data_var[2]^first_mux_data_var[7];first_mux_aD=first_mux_data_var[3]^first_mux_data_var[6];
```

```
                first_mux_InvInput[0]=(!first_mux_data_var[5])^first_mux_aC;
```

```
                first_mux_InvInput[1]=first_mux_data_var[0]^first_mux_aD;
```

```
                first_mux_InvInput[2]=(!first_mux_data_var[7])^first_mux_aB;
```

```
                first_mux_InvInput[3]=first_mux_data_var[2]^first_mux_aA;
```

```
                first_mux_InvInput[4]=first_mux_data_var[1]^first_mux_aD;
```

```
                first_mux_InvInput[5]=first_mux_data_var[4]^first_mux_aC;
```

```
                first_mux_InvInput[6]=first_mux_data_var[3]^first_mux_aA;
```

```
                first_mux_InvInput[7]=first_mux_data_var[6]^first_mux_aB;
```

```
        end
```

```
    default:
```

```
        begin
```

```

    first_mux_InvInput=first_mux_data_var;

end

endcase

//Convert elements from GF(2^8) into two elements of GF(2^4^2)

first_mux_aA=first_mux_InvInput[1]^first_mux_InvInput[7];
first_mux_aB=first_mux_InvInput[5]^first_mux_InvInput[7];
first_mux_aC=first_mux_InvInput[4]^first_mux_InvInput[6];

first_mux_al_t[0]=first_mux_aC^first_mux_InvInput[0]^first_mux_InvInput[5];
first_mux_al_t[1]=first_mux_InvInput[1]^first_mux_InvInput[2];
first_mux_al_t[2]=first_mux_aA;
first_mux_al_t[3]=first_mux_InvInput[2]^first_mux_InvInput[4];

first_mux_ah_t[0]=first_mux_aC^first_mux_InvInput[5];
first_mux_ah_t[1]=first_mux_aA^first_mux_aC;
first_mux_ah_t[2]=first_mux_aB^first_mux_InvInput[2]^first_mux_InvInput[3];
first_mux_ah_t[3]=first_mux_aB;

al = (first_mux_al_t);
ah = (first_mux_ah_t);
next_ah_reg = (first_mux_ah_t);

end

```

```
//end_mux:
```

```
reg[7:0] end_mux_data_var,end_mux_data_o_var;
```

```
reg end_mux_aA,end_mux_aB,end_mux_aC,end_mux_aD;
```

```
always @(decrypt_i or inva)
```

```
begin
```

```
//Take the output of the inverter
```

```
end_mux_data_var=inva;
```

```
case(decrypt_i)
```

```
0:
```

```
begin
```

```
//Apply affine trasformation
```

```
end_mux_aA=end_mux_data_var[0]^end_mux_data_var[1];end_mux_aB=end_mux_data_var[2]^end_mux_data_var[3];
```

```
end_mux_aC=end_mux_data_var[4]^end_mux_data_var[5];end_mux_aD=end_mux_data_var[6]^end_mux_data_var[7];
```

```
end_mux_data_o_var[0]=(!end_mux_data_var[0])^end_mux_aC^end_mux_aD;
```

```
end_mux_data_o_var[1]=(!end_mux_data_var[5])^end_mux_aA^end_mux_aD;
```

```
end_mux_data_o_var[2]=end_mux_data_var[2]^end_mux_aA^end_mux_aD;
```

```
end_mux_data_o_var[3]=end_mux_data_var[7]^end_mux_aA^end_mux_aB;
```

```
end_mux_data_o_var[4]=end_mux_data_var[4]^end_mux_aA^end_mux_aB;
```

```
end_mux_data_o_var[5]=(!end_mux_data_var[1])^end_mux_aB^end_mux_aC;
```

```
end_mux_data_o_var[6]=(!end_mux_data_var[6])^end_mux_aB^end_mux_aC;  
end_mux_data_o_var[7]=end_mux_data_var[3]^end_mux_aC^end_mux_aD;
```

```
data_o = (end_mux_data_o_var);  
end
```

```
default:
```

```
begin
```

```
data_o = (end_mux_data_var);  
end
```

```
endcase
```

```
end
```

```
//inversemap:
```

```
reg[3:0] aA,aB;
```

```
reg[3:0] inversemap_alp_t,inversemap_ahp_t;
```

```
reg[7:0] inversemap_inva_t;
```

```
always @(alp or ahp)
```

```
begin
```

```
inversemap_alp_t=alp;
```

```
inversemap_ahp_t=ahp;
```

```
aA=inversemap_alp_t[1]^inversemap_ahp_t[3];
```

```
aB=inversemap_ahp_t[0]^inversemap_ahp_t[1];
```

```

inversemap_inva_t[0]=inversemap_alp_t[0]^inversemap_ahp_t[0];
inversemap_inva_t[1]=aB^inversemap_ahp_t[3];
inversemap_inva_t[2]=aA^aB;
inversemap_inva_t[3]=aB^inversemap_alp_t[1]^inversemap_ahp_t[2];
inversemap_inva_t[4]=aA^aB^inversemap_alp_t[3];
inversemap_inva_t[5]=aB^inversemap_alp_t[2];
inversemap_inva_t[6]=aA^inversemap_alp_t[2]^inversemap_alp_t[3]^inversemap_ahp_t[0];
inversemap_inva_t[7]=aB^inversemap_alp_t[2]^inversemap_ahp_t[3];

```

```

inva = (inversemap_inva_t);

```

```

end

```

```

//mul1:

```

```

reg[3:0] mul1_alxh_t;
reg[3:0] mul1_aA,mul1_a;

```

```

always @(ah or al)

```

```

begin

```

```

//alxah

```

```

mul1_aA=al[0]^al[3];
mul1_a=al[2]^al[3];

```

```

mul1_alxh_t[0]=(al[0]&ah[0])^(al[3]&ah[1])^(al[2]&ah[2])^(al[1]&ah[3]);
mul1_alxh_t[1]=(al[1]&ah[0])^(mul1_aA&ah[1])^(mul1_a&ah[2])^((al[1]^al[2])&ah[3]);
mul1_alxh_t[2]=(al[2]&ah[0])^(al[1]&ah[1])^(mul1_aA&ah[2])^(mul1_a&ah[3]);
mul1_alxh_t[3]=(al[3]&ah[0])^(al[2]&ah[1])^(al[1]&ah[2])^(mul1_aA&ah[3]);

```

```
alxh = (mul1_alxh_t);
```

```
end
```

```
//mul2:
```

```
reg[3:0] mul2_ahp_t;
```

```
reg[3:0] mul2_aA,mul2_aB;
```

```
always @(d or ah_reg)
```

```
begin
```

```
//ahxd
```

```
mul2_aA=ah_reg[0]^ah_reg[3];
```

```
mul2_aB=ah_reg[2]^ah_reg[3];
```

```
mul2_ahp_t[0]=(ah_reg[0]&d[0])^(ah_reg[3]&d[1])^(ah_reg[2]&d[2])^(ah_reg[1]&d[3]);
```

```
mul2_ahp_t[1]=(ah_reg[1]&d[0])^(mul2_aA&d[1])^(mul2_aB&d[2])^((ah_reg[1]^ah_reg[2])&d[3]);
```

```
mul2_ahp_t[2]=(ah_reg[2]&d[0])^(ah_reg[1]&d[1])^(mul2_aA&d[2])^(mul2_aB&d[3]);
```

```
mul2_ahp_t[3]=(ah_reg[3]&d[0])^(ah_reg[2]&d[1])^(ah_reg[1]&d[2])^(mul2_aA&d[3]);
```

```
ahp = (mul2_ahp_t);
```

```
end
```

```
//mul3:
```



```
reg[3:0] mul3_alp_t;
```

```
reg[3:0] mul3_aA,mul3_aB;
```

```
always @(d or alph)
```

```
begin
```

```
//dxaI
```

```
mul3_aA=d[0]^d[3];
```

```
mul3_aB=d[2]^d[3];
```

```
mul3_alp_t[0]=(d[0]&alph[0])^(d[3]&alph[1])^(d[2]&alph[2])^(d[1]&alph[3]);
```

```
mul3_alp_t[1]=(d[1]&alph[0])^(mul3_aA&alph[1])^(mul3_aB&alph[2])^((d[1]^d[2])&alph[3]);
```

```
mul3_alp_t[2]=(d[2]&alph[0])^(d[1]&alph[1])^(mul3_aA&alph[2])^(mul3_aB&alph[3]);
```

```
mul3_alp_t[3]=(d[3]&alph[0])^(d[2]&alph[1])^(d[1]&alph[2])^(mul3_aA&alph[3]);
```

```
alp = (mul3_alp_t);
```

```
end
```

```
//intermediate:
```

```
reg[3:0] intermediate_aA,intermediate_aB;
```

```
reg[3:0] intermediate_ah2e,intermediate_ah2epl2,intermediate_to_invert_var;
```

```
always @(ah2 or al2 or alxh)
```

```
begin
```

```
//ah square is multiplied with e
```

```

intermediate_aA=ah2[0]^ah2[1];
intermediate_aB=ah2[2]^ah2[3];
intermediate_ah2e[0]=ah2[1]^intermediate_aB;
intermediate_ah2e[1]=intermediate_aA;
intermediate_ah2e[2]=intermediate_aA^ah2[2];
intermediate_ah2e[3]=intermediate_aA^intermediate_aB;

```

```

//Addition of intermediate_ah2e plus al2

```

```

intermediate_ah2epl2[0]=intermediate_ah2e[0]^al2[0];
intermediate_ah2epl2[1]=intermediate_ah2e[1]^al2[1];
intermediate_ah2epl2[2]=intermediate_ah2e[2]^al2[2];
intermediate_ah2epl2[3]=intermediate_ah2e[3]^al2[3];

```

```

//Addition of last result with the result of f(alxah)

```

```

intermediate_to_invert_var[0]=intermediate_ah2epl2[0]^alxh[0];
intermediate_to_invert_var[1]=intermediate_ah2epl2[1]^alxh[1];
intermediate_to_invert_var[2]=intermediate_ah2epl2[2]^alxh[2];
intermediate_to_invert_var[3]=intermediate_ah2epl2[3]^alxh[3];

```

```

//Registers

```

```

next_to_invert = (intermediate_to_invert_var);

```

```

end

```

```

//inversion:

```

```

reg[3:0] inversion_to_invert_var;
reg[3:0] inversion_aA,inversion_d_t;

```

```
always @(to_invert)
```

```
begin
```

```
inversion_to_invert_var=to_invert;
```

```
//Invert the result in GF(2^4)
```

```
inversion_aA=inversion_to_invert_var[1]^inversion_to_invert_var[2]^inversion_to_invert_var[3]^(inversion_to_invert_var[1]&inversion_to_invert_var[2]&inversion_to_invert_var[3]);
```

```
inversion_d_t[0]=inversion_aA^inversion_to_invert_var[0]^(inversion_to_invert_var[0]&inversion_to_invert_var[2])^(inversion_to_invert_var[1]&inversion_to_invert_var[2])^(inversion_to_invert_var[0]&inversion_to_invert_var[1]&inversion_to_invert_var[2]);
```

```
inversion_d_t[1]=(inversion_to_invert_var[0]&inversion_to_invert_var[1])^(inversion_to_invert_var[0]&inversion_to_invert_var[2])^(inversion_to_invert_var[1]&inversion_to_invert_var[2])^inversion_to_invert_var[3]^(inversion_to_invert_var[1]&inversion_to_invert_var[3])^(inversion_to_invert_var[0]&inversion_to_invert_var[1]&inversion_to_invert_var[3]);
```

```
inversion_d_t[2]=(inversion_to_invert_var[0]&inversion_to_invert_var[1])^inversion_to_invert_var[2]^(inversion_to_invert_var[0]&inversion_to_invert_var[2])^inversion_to_invert_var[3]^(inversion_to_invert_var[0]&inversion_to_invert_var[3])^(inversion_to_invert_var[0]&inversion_to_invert_var[2]&inversion_to_invert_var[3]);
```

```
inversion_d_t[3]=inversion_aA^(inversion_to_invert_var[0]&inversion_to_invert_var[3])^(inversion_to_invert_var[1]&inversion_to_invert_var[3])^(inversion_to_invert_var[2]&inversion_to_invert_var[3]);
```

```
d = (inversion_d_t);
```

```
end
```

```
//sum1:
```

```
reg[3:0] sum1_alph_t;
```

```
always @(ah or al)
```

```
begin
```

```
    sum1_alph_t[0]=al[0]^ah[0];
```

```
    sum1_alph_t[1]=al[1]^ah[1];
```

```
    sum1_alph_t[2]=al[2]^ah[2];
```

```
    sum1_alph_t[3]=al[3]^ah[3];
```

```
    next_alph = (sum1_alph_t);
```

```
end
```

```
//square1:
```

```
reg[3:0] square1_ah_t;
```

```
always @(ah)
```

```
begin
```

```
    square1_ah_t[0]=ah[0]^ah[2];
```

```
    square1_ah_t[1]=ah[2];
```

```
    square1_ah_t[2]=ah[1]^ah[3];
```

```
    square1_ah_t[3]=ah[3];
```

```
    ah2 = (square1_ah_t);
```

```
end
```

```
//square2:
reg[3:0] square2_al_t;

always @(al)
begin

    square2_al_t[0]=al[0]^al[2];
    square2_al_t[1]=al[2];
    square2_al_t[2]=al[1]^al[3];
    square2_al_t[3]=al[3];

    al2 = (square2_al_t);

end

endmodule
```