

```

`include "timescale.v"

module aes(clk,reset,load_i,decrypt_i,data_i,key_i,ready_o,data_o);

input clk;

input reset;

input load_i;

input decrypt_i;

input [127:0] data_i;

input [127:0] key_i;

output ready_o;

output [127:0] data_o;


reg ready_o;

reg [127:0] data_o;


reg next_ready_o;

reg this_ready_o; // change


reg keysched_start_i;

reg [3:0] keysched_round_i;

reg [127:0] keysched_last_key_i;

wire [127:0] keysched_new_key_o;

wire keysched_ready_o;

wire keysched_sbox_access_o;

wire [7:0] keysched_sbox_data_o;

wire keysched_sbox_decrypt_o;


reg mixcol_start_i;

reg [127:0] mixcol_data_i;

wire mixcol_ready_o;

```

```
wire [127:0] mixcol_data_o;
```

```
reg subbytes_start_i;
```

```
reg [127:0] subbytes_data_i;
```

```
wire subbytes_ready_o;
```

```
wire [127:0] subbytes_data_o;
```

```
wire [7:0] subbytes_sbox_data_o;
```

```
wire subbytes_sbox_decrypt_o;
```

```
wire [7:0] sbox_data_o;
```

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

```
reg sbox_decrypt_i;
```

```
reg state;
```

```
reg next_state;
```

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

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

```
reg [127:0] addroundkey_data_o;
```

```
reg [127:0] addroundkey_data_o_prev = (0); // NEW SIGNAL
```

```
reg [127:0] next_addroundkey_data_reg;
```

```
reg [127:0] addroundkey_data_reg;
```

```
reg [127:0] addroundkey_data_i;
```

```
reg addroundkey_ready_o;
```

```
reg next_addroundkey_ready_o;
```

```
reg addroundkey_start_i;
```

```
reg next_addroundkey_start_i;
```

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

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

```
reg first_round_reg;
```

```
reg next_first_round_reg;
```

```
sbox sbox1 (.clk(clk), .reset(reset), .data_i(sbox_data_i), .decrypt_i(sbox_decrypt_i),  
.data_o(sbox_data_o));
```

```
subbytes sub1 (.clk(clk), .reset(reset), .start_i(subbytes_start_i), .decrypt_i(decrypt_i),  
.data_i(subbytes_data_i), .ready_o(subbytes_ready_o), .data_o(subbytes_data_o),  
.sbox_data_o(subbytes_sbox_data_o), .sbox_data_i(sbox_data_o),  
.sbox_decrypt_o(subbytes_sbox_decrypt_o));
```

```
mixcolumn mix1 (.clk(clk), .reset(reset), .decrypt_i(decrypt_i), .start_i(mixcol_start_i),  
.data_i(mixcol_data_i), .ready_o(mixcol_ready_o), .data_o(mixcol_data_o));
```

```
keysched ks1 (.clk(clk), .reset(reset), .start_i(keysched_start_i), .round_i(keysched_round_i),  
.last_key_i(keysched_last_key_i), .new_key_o(keysched_new_key_o), .ready_o(keysched_ready_o),  
.sbox_access_o(keysched_sbox_access_o), .sbox_data_o(keysched_sbox_data_o),  
.sbox_data_i(sbox_data_o), .sbox_decrypt_o(keysched_sbox_decrypt_o));
```

```
//registers:
```

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

```
begin
```

```
if(!reset)
```

```
begin
```

```
state = (0);
```

```
ready_o = (0);
```

```
round = (0);
```

```
addroundkey_round = (0);
```

```
addroundkey_data_reg = (0);
```

```
addroundkey_ready_o = (0);
```

```
addroundkey_start_i = (0);
```

```
first_round_reg = (0);
```

```
end
```

```

else
begin

    state = (next_state);

    round = (next_round);
    addroundkey_round = (next_addroundkey_round);
    addroundkey_data_reg = (next_addroundkey_data_reg);
    addroundkey_ready_o = (next_addroundkey_ready_o);
    first_round_reg = (next_first_round_reg);
    addroundkey_start_i = (next_addroundkey_start_i);
    ready_o = (next_ready_o);
end

end

//control:
always @(state or round or addroundkey_data_o or data_i or load_i or decrypt_i or
addroundkey_ready_o or mixcol_ready_o or subbytes_ready_o or subbytes_data_o or mixcol_data_o or
first_round_reg)
begin

    next_state = (state);
    next_round = (round);

    // FROM HERE
    data_o = (addroundkey_data_o);
    if (addroundkey_data_o_prev != (addroundkey_data_o))
begin
        next_ready_o = (1);

```

```

        addroundkey_data_o_prev = (addroundkey_data_o);
end
else
        next_ready_o = (0);
// UNTIL HERE
// CHANGES WERE MADE

//To key schedule module

next_first_round_reg = (0);

subbytes_data_i = (0);
mixcol_data_i = (0);
addroundkey_data_i = (0);
next_addroundkey_start_i = (first_round_reg);
mixcol_start_i = ((addroundkey_ready_o&decrypt_i&round!=10)|(subbytes_ready_o&!decrypt_i));
subbytes_start_i =
((addroundkey_ready_o&!decrypt_i)|(mixcol_ready_o&decrypt_i)|(addroundkey_ready_o&decrypt_i&r
ound==10));

if(decrypt_i&&round!=10)
begin

        addroundkey_data_i = (subbytes_data_o);
        subbytes_data_i = (mixcol_data_o);
        mixcol_data_i = (addroundkey_data_o);

end
else if(!decrypt_i&&round!=0)

```

begin

```
    addroundkey_data_i = (mixcol_data_o);  
    subbytes_data_i = (addroundkey_data_o);  
    mixcol_data_i = (subbytes_data_o);
```

end

else

begin

```
    mixcol_data_i = (subbytes_data_o);  
    subbytes_data_i = (addroundkey_data_o);  
    addroundkey_data_i = (data_i);
```

end

case(state)

0:

begin

if(load_i)

begin

next_state = (1);

if(decrypt_i)

next_round = (10);

else

next_round = (0);

```

    next_first_round_reg = (1);

end

end

1:
begin

    //Counter
    if(!decrypt_i&&mixcol_ready_o)
    begin

        next_addroundkey_start_i = (1);
        addroundkey_data_i = (mixcol_data_o);
        next_round = (round+1);

    end

    else if(decrypt_i&&subbytes_ready_o)
    begin

        next_addroundkey_start_i = (1);
        addroundkey_data_i = (subbytes_data_o);
        next_round = (round-1);

    end

    //Output

```

```
if((round==9&&!decrypt_i) || (round==0&&decrypt_i))
```

```
begin
```

```
    next_addroundkey_start_i = (0);
```

```
    mixcol_start_i = (0);
```

```
    if(subbytes_ready_o)
```

```
    begin
```

```
        addroundkey_data_i = (subbytes_data_o);
```

```
        next_addroundkey_start_i = (1);
```

```
        next_round = (round+1);
```

```
    end
```

```
end
```

```
if((round==10&&!decrypt_i) || (round==0&&decrypt_i))
```

```
begin
```

```
    addroundkey_data_i = (subbytes_data_o);
```

```
    subbytes_start_i = (0);
```

```
    if(addroundkey_ready_o)
```

```
    begin
```

```
        next_ready_o = (1);
```

```
        next_state = (0);
```

```
        next_addroundkey_start_i = (0);
```



```

        next_round = (0);

    end

end

    end

        end

default:
    begin

        next_state = (0);

    end

endcase

end

//addroundkey:
reg[127:0] data_var,round_data_var,round_key_var;

always @(addroundkey_data_i or addroundkey_start_i or addroundkey_data_reg or
addroundkey_round or keysched_new_key_o or keysched_ready_o or key_i or round)
begin

    round_data_var=addroundkey_data_reg;
    next_addroundkey_data_reg = (addroundkey_data_reg);
    next_addroundkey_ready_o = (0);
    next_addroundkey_round = (addroundkey_round);

```

```

addroundkey_data_o = (addroundkey_data_reg);

if(addroundkey_round==1 || addroundkey_round==0)
    keysched_last_key_i = (key_i);
else
    keysched_last_key_i = (keysched_new_key_o);

keysched_start_i = (0);

keysched_round_i = (addroundkey_round);

if(round==0&&addroundkey_start_i)
begin

    //Take the input and xor them with data if round==0;
    data_var=addroundkey_data_i;
    round_key_var=key_i;
    round_data_var=round_key_var^data_var;
    next_addroundkey_data_reg = (round_data_var);
    next_addroundkey_ready_o = (1);

end

else if(addroundkey_start_i&&round!=0)
begin

    keysched_last_key_i = (key_i);
    keysched_start_i = (1);
    keysched_round_i = (1);
    next_addroundkey_round = (1);

```

```
end
else if(addroundkey_round!=round&&keysched_ready_o)
begin
```

```
    next_addroundkey_round = (addroundkey_round+1);
    keysched_last_key_i = (keysched_new_key_o);
    keysched_start_i = (1);
    keysched_round_i = (addroundkey_round+1);
```

```
end
else if(addroundkey_round==round&&keysched_ready_o)
begin
```

```
    data_var=addroundkey_data_i;
    round_key_var=keysched_new_key_o;
    round_data_var=round_key_var^data_var;
    next_addroundkey_data_reg = (round_data_var);
    next_addroundkey_ready_o = (1);
    next_addroundkey_round = (0);
```

```
end
```

```
end
```

```
//sbox_muxes:
```

```
always @(keysched_sbox_access_o or keysched_sbox_decrypt_o or keysched_sbox_data_o or
subbytes_sbox_decrypt_o or subbytes_sbox_data_o)
```

```
begin
```

```
if(keysched_sbox_access_o)
```

```
begin
```

```
    sbox_decrypt_i = (keysched_sbox_decrypt_o);
```

```
    sbox_data_i = (keysched_sbox_data_o);
```

```
end
```

```
else
```

```
begin
```

```
    sbox_decrypt_i = (subbytes_sbox_decrypt_o);
```

```
    sbox_data_i = (subbytes_sbox_data_o);
```

```
end
```

```
end
```

```
endmodule
```