Embedded SW and HW/SW design of of 802.11 MAC layer.

Santosh Krishnan(A53071285) and Adithya Suresh(A53074318).

Working directory:
/home/linux/ieng6/ee260c/skkrishn/project2a/ and /home/linux/ieng6/ee260c/skkrishn/project2b/

1 Times for SW implementation.

	FRAME SIZE (time in ns)				
Parameter	16 bytes(ns)	32 bytes(ns)	48 bytes(ns)	64 bytes(ns)	128 bytes(ns)
Time req. to initialize plain text and other variables	2160	5880	6840	8520	13920
time required for calculating the first CRC for the plaintext frame	24120	45240	71040	87480	171960
Key expansion	170520	170520	170520	170520	170520
Addround key	29280	29280	29280	29280	29280
Sub-bytes	31280	31280	31280	31280	31280
Shift rows	42960	42960	42960	42960	42960
Mixcoll	265560	265560	265560	265560	265560
Addround key	42360	42360	42360	42360	42360
Encryption time per block	3713520	3713520	3713520	3713520	3713520
Appending first CRC	1920	1920	1920	1920	1920
time required for calculating the second CRC for the cypher text frame	29400	50520	76320	92760	177240
Appending second CRC	1920	1920	1920	1920	1920
Encryption time per frame	3884520	7598040	11311560	15025080	29879160
Data Rate(Kbps)	32.9510	33.1613	33.4657	33.6451	33.8425

2. Times for HW/SW implementation.

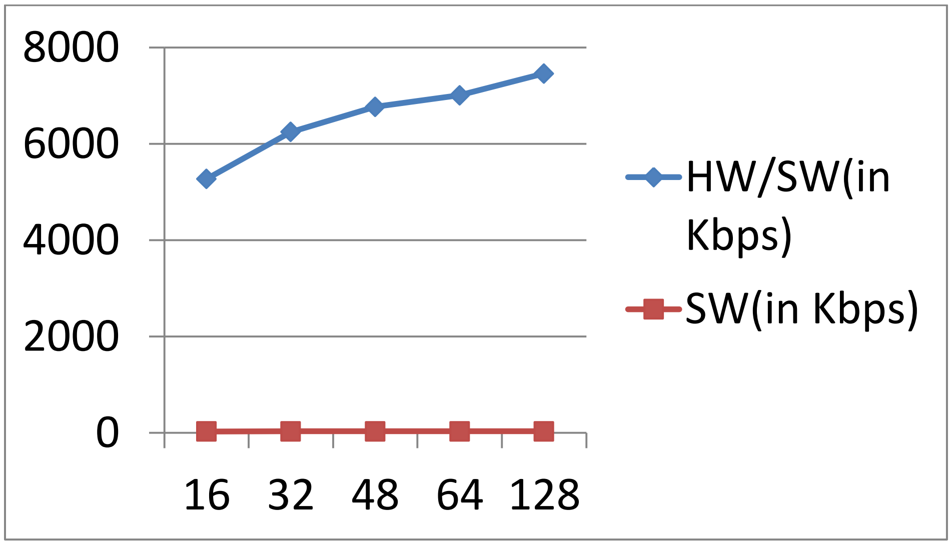
	Frame sizes(time in ns)				
paramter	16	32	48	64	128
Time req. to initialize plain text and other variables and start the code	12846	20814	28206	35742	65550
Encryption time per frame	4658	9144	13298	17856	35352
Appending first CRC	478	478	478	478	478
time required for calculating the second CRC for the cypher text frame	5880	10104	14408	18552	35448
Append second crc	478	478	478	478	478
Data rate(Mbps):	5.2740	6.2515	6.7703	7.0100	7.4615

3 Changes made to the Verilog file:

Apart from the changes suggested, the following changes were made to the Verilog code,

- Changing the states where reading from and writing to the DPSRAM was taking place to ensure that write and read cycle times were only one cycle. Avoided hardcoding of the read and write addresses, used 3, 32 bit adders, to calculate the port addresses

- Minimizing number of states to perform encryption to ensure that number of cycles to perform encryption decreases.
- Effect of changes:
  - Before changes: Cycle time=12.4ns, number of cycles=83, Area ~15200um^2
  - After changes: Cycle time=6.5ns, number of cycles=51, Area ~30000um^2
  - Effect of change: more than 50% improvement in area delay product.



Figre 1: Plot of the data rate vs the frame size in bytes for HW/SW and SW

4. Results:

When we implement the 802.11 MAC in software, we see that the time taken to perform AES encryption is extremely large thus resulting in very small Data rates. (figure 1) This is the reason why we go for a mix of HW and SW implementation with the aim of reducing the large AES encryption time. By going for a mix of HW and SW, we not only get the flexibility of SW but also the performance of HW for key parts of the 802.11 MAC. This can be seen in the times of the HW/SW implementation. For both HW/SW and SW, we can get better performance by increasing the frame size since the SW overhead (initialization, reading, writing, etc.) per byte decreases. For the HW/SW implementation, if we do a static timing analysis for the different paths, we see that we see that the most critical task is that which is involved with CRC calculation. So if we can take this task to a HW implementation, we can get additional benefits.

5. Appendix A:

The C Code

```
#include <stdio.h>

#include "crc_table.h"

//Initializing the key; The four words of the key are listed below.

unsigned long key_0 = 0x2b7e1516;

unsigned long key_1 = 0x28aed2a6;

unsigned long key_2 = 0xabf71588;

unsigned long key_3 = 0x09cf4f3c;

unsigned long get_crc(int no_of_bytes, unsigned char data[])

{
```

```
            unsigned long crc_temp = 0xFFFFFFFF;

            int i, index;


            index = ((int)(crc_temp >> 24)) & 0xFF;

            crc_temp = (crc_temp << 8) ^ crc_table[index];


            for (i = 0; i < no_of_bytes; i++)

            {

                        index = ((int)(crc_temp >> 24) ^ data[i]) & 0xFF;

                        crc_temp = (crc_temp << 8) ^ crc_table[index];

            }

            crc_temp = crc_temp ^ 0xFFFFFFFF;


            return crc_temp;

}

//Main function calls the key_expansion and encryption transformation functions to encrypt input data with the given
key

int TestMain()

{

            //Initializing plaintext

  unsigned char plaintext[16]={0x32, 0x43, 0xf6, 0xa8, 0x88, 0x5a, 0x30, 0x8d, 0x31, 0x31, 0x98, 0xa2, 0xe0, 0x37, 0x07,
0x34};

    unsigned char ciphertext[24];

            int no_of_bytes=16;

      int no_of_blocks=no_of_bytes/16;

            unsigned long crc;

            unsigned long p;

            unsigned long *mem_addr;

            unsigned long addr=0x01000100;

            unsigned long plain_addr;

            unsigned long c_addr;

            unsigned long done;
```

```
	unsigned long temp;

	//Initializing round counter to zero

	int round_cnt=0;


	//Other variables needed for loop control

	int i, j, k;


	plain_addr=addr;


	//Determining the CRC of plaintext before encryption

	crc=get_crc(no_of_bytes, plaintext);


	//Write the plaintext to the dpsram starting from location 0x01000100

	for(i=0;i<no_of_bytes;i=i+4)

	{

		mem_addr = (unsigned long *) addr;


	p=(((plaintext[i+3]&0x000000ff)<<24)|((plaintext[i+2]&0x000000ff)<<16)|((plaintext[i+1]&0x000000ff)<<8)|(plai
ntext[i+0]&0x000000ff));

		*(mem_addr) =p;

		addr=addr+0x00000004;

	}

	//Write the starting location in dpsram to location 0x01000008

	addr=0x01000008;

	mem_addr = (unsigned long *) addr;

	*(mem_addr) = plain_addr;


	//write the key_0 to the location 0x0100000c

	addr=0x0100000c;

	mem_addr = (unsigned long *) addr;

	*(mem_addr) = key_0;
```

```
//write the key_1 to the location 0x01000010

addr=0x01000010;

mem_addr = (unsigned long *) addr;

*(mem_addr) = key_1;


//write the key_2 to the location 0x01000014

addr=0x01000014;

mem_addr = (unsigned long *) addr;

*(mem_addr) = key_2;


//write the key_3 to the location 0x01000018

addr=0x01000018;

mem_addr = (unsigned long *) addr;

*(mem_addr) = key_3;


//write 0 to done register

addr=0x0100001C;

mem_addr = (unsigned long *) addr;

*(mem_addr) = 0x00000000;


//write the required data to start_encrypt to start the encryprtion

addr=0x01000004;

mem_addr = (unsigned long *) addr;

*(mem_addr) = 0x01001001;


//write the required data to start_encrypt to continue the encryprtion

addr=0x01000004;

mem_addr = (unsigned long *) addr;

*(mem_addr) = 0x01001000;


while(1)
```

```
{
        addr=0x0100001C;
        mem_addr = (unsigned long *) addr;
        done=*(unsigned long*)mem_addr;
        if(done==0x00000001)
        {
                break;
        }
}
c_addr=0x01000200;
for(k=0; k<no_of_blocks; k++)
{
        for (j=0; j<4; j++)
{
                mem_addr = (unsigned long *) c_addr;
                temp=*(mem_addr);
                ciphertext[(k*16)+(3+(j*4))]=(unsigned char) ((temp & 0xff000000)>>24);
                ciphertext[(k*16)+(2+(j*4))]=(unsigned char) ((temp & 0x00ff0000)>>16);
                ciphertext[(k*16)+(1+(j*4))]=(unsigned char) ((temp & 0x0000ff00)>>8);
                ciphertext[(k*16)+(0+(j*4))]=(unsigned char) ((temp & 0x000000ff)>>0);
                c_addr=c_addr+0x00000004;
}
}
// Append the CRC to the ciphertext
ciphertext[no_of_bytes]   = (unsigned char)(crc >> 24);
ciphertext[no_of_bytes+1] = (unsigned char)(crc >> 16);
ciphertext[no_of_bytes+2] = (unsigned char)(crc >> 8);
ciphertext[no_of_bytes+3] = (unsigned char)(crc);

// Compute the CRC of the whole payload
crc = get_crc(no_of_bytes+4, ciphertext);
```

```
        // Append the CRC to the payload
        ciphertext[no_of_bytes+4]   = (unsigned char)(crc >> 24);
        ciphertext[no_of_bytes+4+1] = (unsigned char)(crc >> 16);
        ciphertext[no_of_bytes+4+2] = (unsigned char)(crc >> 8);
        ciphertext[no_of_bytes+4+3] = (unsigned char)(crc);
}
```

6. Appendix B

The Verilog file:

```
module AES_128bit(
clk,
nreset,
start_encrypt,
plain_addr,
//cipher_addr_off,
//frame_size,
key_3,
key_2,
key_1,
key_0,
port_A_clk,
port_A_data_in,
port_A_data_out,
port_A_addr,
port_A_we,
done
);
input clk;
input nreset;
// Initializes the AES_128bit module
input [31:0] start_encrypt;
```

```
// Tells AES_128bit to start encrypting the given frame

input [31:0] plain_addr;

// Starting address of the plaintext frame

// i.e., specifies from where AES_128bit must read the plaintext frame

//input [15:0] cipher_addr_off;

// Offset added to starting address of plaintext frame to obtain starting address of cipher text

//input [7:0] frame_size;

// Length of the frame in bytes

input [31:0] key_0;

// Contains the least significant word of the 4 word (128 bit) key

input [31:0] key_1;

// Contains the second word of the 4 word (128 bit) key

input [31:0] key_2;

// Contains the third word of the 4 word (128 bit) key

input [31:0] key_3;

// Contains the most significant word of the 4 word (128 bit) key

input [31:0] port_A_data_out;

// read data from the dpsram (plaintext)

output [31:0] port_A_data_in;

// write data to the dpsram (ciphertext)

output [31:0] port_A_addr;

// address of dpsram being read/written

output port_A_clk;

// clock to dpsram (drive this with the input clk)

output port_A_we;

// read/write selector for dpsram

output done;

// done is a signal to indicate that encryption of the frame is complete

assign port_A_clk=clk;

reg done;

reg port_A_we;
reg [31:0] port_A_data_in;

reg [7:0] pt_trans00;

reg [7:0] pt_trans01;

reg [7:0] pt_trans02;

reg [7:0] pt_trans03;

reg [7:0] pt_trans10;

reg [7:0] pt_trans11;

reg [7:0] pt_trans12;

reg [7:0] pt_trans13;

reg [7:0] pt_trans20;

reg [7:0] pt_trans21;

reg [7:0] pt_trans22;

reg [7:0] pt_trans23;

reg [7:0] pt_trans30;

reg [7:0] pt_trans31;

reg [7:0] pt_trans32;

reg [7:0] pt_trans33;

reg [7:0] cpt_trans00;

reg [7:0] cpt_trans01;

reg [7:0] cpt_trans02;

reg [7:0] cpt_trans03;

reg [7:0] cpt_trans10;

reg [7:0] cpt_trans11;

reg [7:0] cpt_trans12;

reg [7:0] cpt_trans13;

reg [7:0] cpt_trans20;

reg [7:0] cpt_trans21;

reg [7:0] cpt_trans22;

reg [7:0] cpt_trans23;

reg [7:0] cpt_trans30;

reg [7:0] cpt_trans31;
reg [7:0] cpt_trans32;

reg [7:0] cpt_trans33;

reg [7:0] pt_trans0;

reg [7:0] pt_trans1;

reg [7:0] pt_trans2;

reg [7:0] pt_trans3;

reg [7:0] rcon[9:0];

reg f;

reg [7:0] ct00;

reg [7:0] ct01;

reg [7:0] ct02;

reg [7:0] ct03;

reg [7:0] ct10;

reg [7:0] ct11;

reg [7:0] ct12;

reg [7:0] ct13;

reg [7:0] ct20;

reg [7:0] ct21;

reg [7:0] ct22;

reg [7:0] ct23;

reg [7:0] ct30;

reg [7:0] ct31;

reg [7:0] ct32;

reg [7:0] ct33;

reg [7:0] entries[15:0][15:0];

reg [3:0] cnt_rk;

reg [7:0] tt0;

reg [7:0] tt1;

reg [7:0] tt2;

reg [7:0] tt3;

reg [7:0] temp00;
reg [7:0] temp01;

reg [7:0] temp02;

reg [7:0] temp03;

reg [7:0] temp10;

reg [7:0] temp11;

reg [7:0] temp12;

reg [7:0] temp13;

reg [7:0] temp20;

reg [7:0] temp21;

reg [7:0] temp22;

reg [7:0] temp23;

reg [7:0] temp30;

reg [7:0] temp31;

reg [7:0] temp32;

reg [7:0] temp33;

reg [1:0] status;

reg [31:0] w0,w1,w2,w3,text_in_r;

reg [1:0] lcnt;

reg [1:0] check;

reg [15:0] cypher_offset;

reg no;

wire p;

wire [31:0] port_A_addr_w,port_A_addr_r;

wire [4:0] g,ff;

assign p=(cnt_rk==0|cnt_rk==10)?1'b1:1'b0;

assign g=((lcnt==0)?4'h0:((lcnt==1)?4'h4:((lcnt==2)?4'h8:4'hc)));

assign port_A_addr_r = (lcnt==0)?plain_addr:plain_addr+g;

assign ff=((check==0)?4'hc:((check==1)?4'h0:((check==2)?4'h4:4'h8)));

assign port_A_addr_w = plain_addr+cypher_offset+ff;

assign port_A_addr = (port_A_we==1'b1)?port_A_addr_w:port_A_addr_r;

reg [5:0] state;
reg [3:0] blk_no,blk;

always @ (posedge clk)

begin

        if(nreset==1'b0)

        begin

        entries[0][0]=8'h63;

        entries[1][0]=8'hca;

        entries[2][0]=8'hb7;

        entries[3][0]=8'h04;

        entries[4][0]=8'h09;

        entries[5][0]=8'h53;

        entries[6][0]=8'hd0;

        entries[7][0]=8'h51;

        entries[8][0]=8'hcd;

        entries[9][0]=8'h60;

        entries[10][0]=8'he0;

        entries[11][0]=8'he7;

        entries[12][0]=8'hba;

        entries[13][0]=8'h70;

        entries[14][0]=8'he1;

        entries[15][0]=8'h8c;


        entries[0][1]=8'h7c;

        entries[1][1]=8'h82;

        entries[2][1]=8'hfd;

        entries[3][1]=8'hc7;

        entries[4][1]=8'h83;

        entries[5][1]=8'hd1;

        entries[6][1]=8'hef;

        entries[7][1]=8'ha3;

        entries[8][1]=8'h0c;
```

```
entries[9][1]=8'h81;

entries[10][1]=8'h32;

entries[11][1]=8'hc8;

entries[12][1]=8'h78;

entries[13][1]=8'h3e;

entries[14][1]=8'hf8;

entries[15][1]=8'ha1;


entries[0][2]=8'h77;

entries[1][2]=8'hc9;

entries[2][2]=8'h93;

entries[3][2]=8'h23;

entries[4][2]=8'h2c;

entries[5][2]=8'h00;

entries[6][2]=8'haa;

entries[7][2]=8'h40;

entries[8][2]=8'h13;

entries[9][2]=8'h4f;

entries[10][2]=8'h3a;

entries[11][2]=8'h37;

entries[12][2]=8'h25;

entries[13][2]=8'hb5;

entries[14][2]=8'h98;

entries[15][2]=8'h89;


entries[0][3]=8'h7b;

entries[1][3]=8'h7d;

entries[2][3]=8'h26;

entries[3][3]=8'hc3;

entries[4][3]=8'h1a;

entries[5][3]=8'hed;
```

```
entries[6][3]=8'hfb;

entries[7][3]=8'h8f;

entries[8][3]=8'hec;

entries[9][3]=8'hdc;

entries[10][3]=8'h0a;

entries[11][3]=8'h6d;

entries[12][3]=8'h2e;

entries[13][3]=8'h66;

entries[14][3]=8'h11;

entries[15][3]=8'h0d;


entries[0][4]=8'hf2;

entries[1][4]=8'hfa;

entries[2][4]=8'h36;

entries[3][4]=8'h18;

entries[4][4]=8'h1b;

entries[5][4]=8'h20;

entries[6][4]=8'h43;

entries[7][4]=8'h92;

entries[8][4]=8'h5f;

entries[9][4]=8'h22;

entries[10][4]=8'h49;

entries[11][4]=8'h8d;

entries[12][4]=8'h1c;

entries[13][4]=8'h48;

entries[14][4]=8'h69;

entries[15][4]=8'hbf;


entries[0][5]=8'h6b;

entries[1][5]=8'h59;

entries[2][5]=8'h3f;
```

```
entries[3][5]=8'h96;

entries[4][5]=8'h6e;

entries[5][5]=8'hfc;

entries[6][5]=8'h4d;

entries[7][5]=8'h9d;

entries[8][5]=8'h97;

entries[9][5]=8'h2a;

entries[10][5]=8'h06;

entries[11][5]=8'hd5;

entries[12][5]=8'ha6;

entries[13][5]=8'h03;

entries[14][5]=8'hd9;

entries[15][5]=8'he6;


entries[0][6]=8'h6f;

entries[1][6]=8'h47;

entries[2][6]=8'hf7;

entries[3][6]=8'h05;

entries[4][6]=8'h5a;

entries[5][6]=8'hb1;

entries[6][6]=8'h33;

entries[7][6]=8'h38;

entries[8][6]=8'h44;

entries[9][6]=8'h90;

entries[10][6]=8'h24;

entries[11][6]=8'h4e;

entries[12][6]=8'hb4;

entries[13][6]=8'hf6;

entries[14][6]=8'h8e;

entries[15][6]=8'h42;
```

```
entries[0][7]=8'hc5;

entries[1][7]=8'hf0;

entries[2][7]=8'hcc;

entries[3][7]=8'h9a;

entries[4][7]=8'ha0;

entries[5][7]=8'h5b;

entries[6][7]=8'h85;

entries[7][7]=8'hf5;

entries[8][7]=8'h17;

entries[9][7]=8'h88;

entries[10][7]=8'h5c;

entries[11][7]=8'ha9;

entries[12][7]=8'hc6;

entries[13][7]=8'h0e;

entries[14][7]=8'h94;

entries[15][7]=8'h68;


entries[0][8]=8'h30;

entries[1][8]=8'had;

entries[2][8]=8'h34;

entries[3][8]=8'h07;

entries[4][8]=8'h52;

entries[5][8]=8'h6a;

entries[6][8]=8'h45;

entries[7][8]=8'hbc;

entries[8][8]=8'hc4;

entries[9][8]=8'h46;

entries[10][8]=8'hc2;

entries[11][8]=8'h6c;

entries[12][8]=8'he8;

entries[13][8]=8'h61;
```

```
entries[14][8]=8'h9b;

entries[15][8]=8'h41;


entries[0][9]=8'h01;

entries[1][9]=8'hd4;

entries[2][9]=8'ha5;

entries[3][9]=8'h12;

entries[4][9]=8'h3b;

entries[5][9]=8'hcb;

entries[6][9]=8'hf9;

entries[7][9]=8'hb6;

entries[8][9]=8'ha7;

entries[9][9]=8'hee;

entries[10][9]=8'hd3;

entries[11][9]=8'h56;

entries[12][9]=8'hdd;

entries[13][9]=8'h35;

entries[14][9]=8'h1e;

entries[15][9]=8'h99;


entries[0][10]=8'h67;

entries[1][10]=8'ha2;

entries[2][10]=8'he5;

entries[3][10]=8'h80;

entries[4][10]=8'hd6;

entries[5][10]=8'hbe;

entries[6][10]=8'h02;

entries[7][10]=8'hda;

entries[8][10]=8'h7e;

entries[9][10]=8'hb8;

entries[10][10]=8'hac;
```

```
entries[11][10]=8'hf4;

entries[12][10]=8'h74;

entries[13][10]=8'h57;

entries[14][10]=8'h87;

entries[15][10]=8'h2d;


entries[0][11]=8'h2b;

entries[1][11]=8'haf;

entries[2][11]=8'hf1;

entries[3][11]=8'he2;

entries[4][11]=8'hb3;

entries[5][11]=8'h39;

entries[6][11]=8'h7f;

entries[7][11]=8'h21;

entries[8][11]=8'h3d;

entries[9][11]=8'h14;

entries[10][11]=8'h62;

entries[11][11]=8'hea;

entries[12][11]=8'h1f;

entries[13][11]=8'hb9;

entries[14][11]=8'he9;

entries[15][11]=8'h0f;


entries[0][12]=8'hfe;

entries[1][12]=8'h9c;

entries[2][12]=8'h71;

entries[3][12]=8'heb;

entries[4][12]=8'h29;

entries[5][12]=8'h4a;

entries[6][12]=8'h50;

entries[7][12]=8'h10;
```

```
entries[8][12]=8'h64;

entries[9][12]=8'hde;

entries[10][12]=8'h91;

entries[11][12]=8'h65;

entries[12][12]=8'h4b;

entries[13][12]=8'h86;

entries[14][12]=8'hce;

entries[15][12]=8'hb0;


entries[0][13]=8'hd7;

entries[1][13]=8'ha4;

entries[2][13]=8'hd8;

entries[3][13]=8'h27;

entries[4][13]=8'he3;

entries[5][13]=8'h4c;

entries[6][13]=8'h3c;

entries[7][13]=8'hff;

entries[8][13]=8'h5d;

entries[9][13]=8'h5e;

entries[10][13]=8'h95;

entries[11][13]=8'h7a;

entries[12][13]=8'hbd;

entries[13][13]=8'hc1;

entries[14][13]=8'h55;

entries[15][13]=8'h54;


entries[0][14]=8'hab;

entries[1][14]=8'h72;

entries[2][14]=8'h31;

entries[3][14]=8'hb2;

entries[4][14]=8'h2f;
```

```
entries[5][14]=8'h58;

entries[6][14]=8'h9f;

entries[7][14]=8'hf3;

entries[8][14]=8'h19;

entries[9][14]=8'h0b;

entries[10][14]=8'he4;

entries[11][14]=8'hae;

entries[12][14]=8'h8b;

entries[13][14]=8'h1d;

entries[14][14]=8'h28;

entries[15][14]=8'hbb;


entries[0][15]=8'h76;

entries[1][15]=8'hc0;

entries[2][15]=8'h15;

entries[3][15]=8'h75;

entries[4][15]=8'h84;

entries[5][15]=8'hcf;

entries[6][15]=8'ha8;

entries[7][15]=8'hd2;

entries[8][15]=8'h73;

entries[9][15]=8'hdb;

entries[10][15]=8'h79;

entries[11][15]=8'h08;

entries[12][15]=8'h8a;

entries[13][15]=8'h9e;

entries[14][15]=8'hdf;

entries[15][15]=8'h16;


check=2'b00;

tt0=8'h00;
```

```
tt1=8'h00;

tt2=8'h00;

tt3=8'h00;

rcon[0]=8'h01;

rcon[1]=8'h02;

rcon[2]=8'h04;

rcon[3]=8'h08;

rcon[4]=8'h10;

rcon[5]=8'h20;

rcon[6]=8'h40;

rcon[7]=8'h80;

rcon[8]=8'h1b;

rcon[9]=8'h36;

port_A_data_in=16'h0000;

port_A_we=1'b0;

f=1'b0;

done=1'b0;

cnt_rk=8'h00;

lcnt=2'b00;

status=2'b00;

state=1;

end

else

begin

case(state)

1:begin

        blk_no=start_encrypt[15:12];

        blk=blk_no;

        cnt_rk = 4'h0;

        done=0;

        port_A_we=1'b0;
```

```
		if (start_encrypt[0])
		begin
				f=~f;
			state = 2;
			done = 0;
				lcnt = 1;
		end
		else begin
				status=2'b10;
			state = 1;
		end
end
2:begin
		port_A_we = 0;
		case(lcnt)
		1:begin
				{{cpt_trans30},{cpt_trans20},{cpt_trans10},{cpt_trans00}}= port_A_data_out[31:0];
				state=2;
		end
		2:begin
				{{cpt_trans31},{cpt_trans21},{cpt_trans11},{cpt_trans01}} = port_A_data_out[31:0];
				state=2;
		end
		3:begin
				{{cpt_trans32},{cpt_trans22},{cpt_trans12},{cpt_trans02}} = port_A_data_out[31:0];
				state=2;
		end
		0:begin
				{{cpt_trans33},{cpt_trans23},{cpt_trans13},{cpt_trans03}} = port_A_data_out[31:0];
				state = 16;
				cnt_rk=0;
```

```
        end
        endcase
        if(lcnt==0)
        begin
                lcnt=0;
        end
        else
        begin
                lcnt = lcnt + 1;
        end
end
16:begin
        port_A_we = 0;
        temp00=cpt_trans00;
        temp10=cpt_trans10;
        temp20=cpt_trans20;
        temp30=cpt_trans30;
        temp01=cpt_trans01;
        temp11=cpt_trans11;
        temp21=cpt_trans21;
        temp31=cpt_trans31;
        temp02=cpt_trans02;
        temp12=cpt_trans12;
        temp22=cpt_trans22;
        temp32=cpt_trans32;
        temp03=cpt_trans03;
        temp13=cpt_trans13;
        temp23=cpt_trans23;
        temp33=cpt_trans33;
        pt_trans00=cpt_trans00;
        pt_trans10=cpt_trans10;
```

```
pt_trans20=cpt_trans20;
pt_trans30=cpt_trans30;
pt_trans01=cpt_trans01;
pt_trans11=cpt_trans11;
pt_trans21=cpt_trans21;
pt_trans31=cpt_trans31;
pt_trans02=cpt_trans02;
pt_trans12=cpt_trans12;
pt_trans22=cpt_trans22;
pt_trans32=cpt_trans32;
pt_trans03=cpt_trans03;
pt_trans13=cpt_trans13;
pt_trans23=cpt_trans23;
pt_trans33=cpt_trans33;
ct30=key_0[7:0];
ct20=key_0[15:8];
ct10=key_0[23:16];
ct00=key_0[31:24];
ct31=key_1[7:0];
ct21=key_1[15:8];
ct11=key_1[23:16];
ct01=key_1[31:24];
ct32=key_2[7:0];
ct22=key_2[15:8];
ct12=key_2[23:16];
ct02=key_2[31:24];
ct33=key_3[7:0];
ct23=key_3[15:8];
ct13=key_3[23:16];
ct03=key_3[31:24];
cnt_rk=0;
```

```
          if(blk_no==blk)
                    cypher_offset=start_encrypt[31:16];
          else
                    cypher_offset=cypher_offset+16'h0010;
          state=4;


end
4:begin
          if(cnt_rk!=0)
          begin
                    tt0=entries[ct13[7:4]][ct13[3:0]];
                    tt1=entries[ct23[7:4]][ct23[3:0]];
                    tt2=entries[ct33[7:4]][ct33[3:0]];
                    tt3=entries[ct03[7:4]][ct03[3:0]];
                    temp00=entries[pt_trans00[7:4]][pt_trans00[3:0]];
                    temp01=entries[pt_trans01[7:4]][pt_trans01[3:0]];
                    temp02=entries[pt_trans02[7:4]][pt_trans02[3:0]];
                    temp03=entries[pt_trans03[7:4]][pt_trans03[3:0]];
                    temp10=entries[pt_trans11[7:4]][pt_trans11[3:0]];
                    temp11=entries[pt_trans12[7:4]][pt_trans12[3:0]];
                    temp12=entries[pt_trans13[7:4]][pt_trans13[3:0]];
                    temp13=entries[pt_trans10[7:4]][pt_trans10[3:0]];
                    temp20=entries[pt_trans22[7:4]][pt_trans22[3:0]];
                    temp21=entries[pt_trans23[7:4]][pt_trans23[3:0]];
                    temp22=entries[pt_trans20[7:4]][pt_trans20[3:0]];
                    temp23=entries[pt_trans21[7:4]][pt_trans21[3:0]];
                    temp30=entries[pt_trans33[7:4]][pt_trans33[3:0]];
                    temp31=entries[pt_trans30[7:4]][pt_trans30[3:0]];
                    temp32=entries[pt_trans31[7:4]][pt_trans31[3:0]];
                    temp33=entries[pt_trans32[7:4]][pt_trans32[3:0]];
                    ct00=ct00^tt0^rcon[cnt_rk-1];
                    ct10=ct10^tt1;

                    ct20=ct20^tt2;

                    ct30=ct30^tt3;

                    ct01=ct01^ct00;

                    ct11=ct11^ct10;

                    ct21=ct21^ct20;

                    ct31=ct31^ct30;

                    ct02=ct02^ct01;

                    ct12=ct12^ct11;

                    ct22=ct22^ct21;

                    ct32=ct32^ct31;

                    ct03=ct03^ct02;

                    ct13=ct13^ct12;

                    ct23=ct23^ct22;

                    ct33=ct33^ct32;

               end

               if(p==1'b0)

               begin


          pt_trans00=((mixcoll(temp00,8'h02))^(mixcoll(temp10,8'h03))^(mixcoll(temp20,8'h01))^(mixcoll(temp30,8'h01)
))^ct00;

          pt_trans01=((mixcoll(temp01,8'h02))^(mixcoll(temp11,8'h03))^(mixcoll(temp21,8'h01))^(mixcoll(temp31,8'h01)
))^ct01;
          pt_trans02=((mixcoll(temp02,8'h02))^(mixcoll(temp12,8'h03))^(mixcoll(temp22,8'h01))^(mixcoll(temp32,8'h01)
))^ct02;
          pt_trans03=((mixcoll(temp03,8'h02))^(mixcoll(temp13,8'h03))^(mixcoll(temp23,8'h01))^(mixcoll(temp33,8'h01)
))^ct03;
          pt_trans10=((mixcoll(temp00,8'h01))^(mixcoll(temp10,8'h02))^(mixcoll(temp20,8'h03))^(mixcoll(temp30,8'h01)
))^ct10;
          pt_trans11=((mixcoll(temp01,8'h01))^(mixcoll(temp11,8'h02))^(mixcoll(temp21,8'h03))^(mixcoll(temp31,8'h01)
))^ct11;
          pt_trans12=((mixcoll(temp02,8'h01))^(mixcoll(temp12,8'h02))^(mixcoll(temp22,8'h03))^(mixcoll(temp32,8'h01)
))^ct12;
          pt_trans13=((mixcoll(temp03,8'h01))^(mixcoll(temp13,8'h02))^(mixcoll(temp23,8'h03))^(mixcoll(temp33,8'h01)
))^ct13;
          pt_trans20=((mixcoll(temp00,8'h01))^(mixcoll(temp10,8'h01))^(mixcoll(temp20,8'h02))^(mixcoll(temp30,8'h03)
))^ct20;
          pt_trans21=((mixcoll(temp01,8'h01))^(mixcoll(temp11,8'h01))^(mixcoll(temp21,8'h02))^(mixcoll(temp31,8'h03)
))^ct21;
```

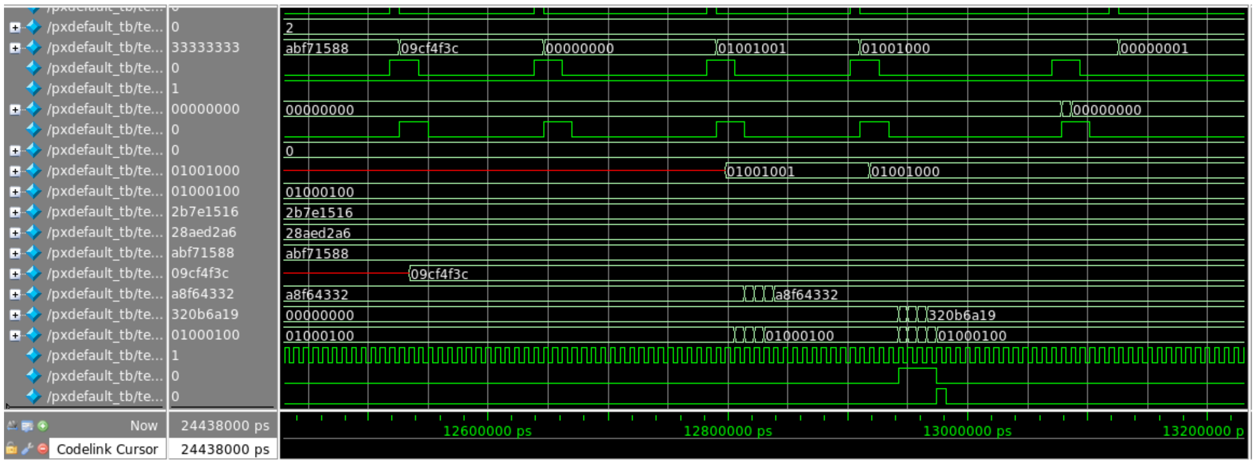
```
            pt_trans22=((mixcoll(temp02,8'h01))^(mixcoll(temp12,8'h01))^(mixcoll(temp22,8'h02))^(mixcoll(temp32,8'h03)
))^ct22;
            pt_trans23=((mixcoll(temp03,8'h01))^(mixcoll(temp13,8'h01))^(mixcoll(temp23,8'h02))^(mixcoll(temp33,8'h03)
))^ct23;
            pt_trans30=((mixcoll(temp00,8'h03))^(mixcoll(temp10,8'h01))^(mixcoll(temp20,8'h01))^(mixcoll(temp30,8'h02)
))^ct30;
            pt_trans31=((mixcoll(temp01,8'h03))^(mixcoll(temp11,8'h01))^(mixcoll(temp21,8'h01))^(mixcoll(temp31,8'h02)
))^ct31;
            pt_trans32=((mixcoll(temp02,8'h03))^(mixcoll(temp12,8'h01))^(mixcoll(temp22,8'h01))^(mixcoll(temp32,8'h02)
))^ct32;
            pt_trans33=((mixcoll(temp03,8'h03))^(mixcoll(temp13,8'h01))^(mixcoll(temp23,8'h01))^(mixcoll(temp33,8'h02)
))^ct33;

                  end

                  else

                  begin

                        pt_trans00=temp00^ct00;//1&2

                        pt_trans01=temp01^ct01;

                        pt_trans02=temp02^ct02;

                        pt_trans03=temp03^ct03;

                        pt_trans10=temp10^ct10;

                        pt_trans11=temp11^ct11;

                        pt_trans12=temp12^ct12;

                        pt_trans13=temp13^ct13;

                        pt_trans20=temp20^ct20;

                        pt_trans21=temp21^ct21;

                        pt_trans22=temp22^ct22;

                        pt_trans23=temp23^ct23;

                        pt_trans30=temp30^ct30;

                        pt_trans31=temp31^ct31;

                        pt_trans32=temp32^ct32;

                        pt_trans33=temp33^ct33;

                  end

                  cnt_rk=cnt_rk+1;

                  state=(cnt_rk==11)?8:4;

                  check=2'b00;
```

```
end
8: begin
          port_A_we = 1;
          case(check)
          0:        begin
          port_A_data_in={{pt_trans30},{pt_trans20},{pt_trans10},{pt_trans00}};
          state = 8;
          end
          1:        begin
          port_A_data_in[31:0] = {{pt_trans31},{pt_trans21},{pt_trans11},{pt_trans01}};
          state = 8;
          end
          2:        begin
          port_A_data_in[31:0] = {{pt_trans32},{pt_trans22},{pt_trans12},{pt_trans02}};
          state = 8;
          end
          3:        begin
          port_A_data_in[31:0] = {{pt_trans33},{pt_trans23},{pt_trans13},{pt_trans03}};
          blk_no=blk_no-1;
                    if(blk_no==4'h0)
                    begin
                              done=0;
                              state=32;
                    end
                    else
                    begin
                              state=16;
                    end
          end
          endcase
          check=check+1;
```

```
        end
        32: begin
                port_A_we = 0;
                done=1;
                state=1;
        end
        default: state=1;
        endcase
        end
end
function [7:0] mixcoll;
  input [7:0] ip_to_be_mixed;
  input [7:0] mul_matrix_entry;
  reg [7:0] r_temp,r_temp2,r_temp3;
  begin
        r_temp=8'h00;
        r_temp2=8'h00;
        r_temp3=8'h00;
        if(mul_matrix_entry==8'h01) begin
                r_temp=ip_to_be_mixed;
        end
        else begin
                r_temp2=(ip_to_be_mixed & 8'h80);
                if(r_temp2==8'h80)
                begin
                        r_temp3=(ip_to_be_mixed<<1)^8'h1b;
                end
                else
                begin
                        r_temp3=ip_to_be_mixed<<1;
                end
                if(mul_matrix_entry==8'h02)
                begin
                        r_temp=r_temp3;
                end
                else
                begin
                        r_temp=r_temp3^ip_to_be_mixed;
                end
        end
        mixcoll=r_temp;
        end
endfunction
endmodule
```

7. Appendix C:

The correct output for frame size =16 bytes;



```
/pxdefault_tb/te... 0
/pxdefault_tb/te... 33333333
/pxdefault_tb/te... 0
/pxdefault_tb/te... 1
/pxdefault_tb/te... 00000000
/pxdefault_tb/te... 0
/pxdefault_tb/te... 0
/pxdefault_tb/te... 01001000
/pxdefault_tb/te... 01000100
/pxdefault_tb/te... 2b7e1516
/pxdefault_tb/te... 28aed2a6
/pxdefault_tb/te... abf71588
/pxdefault_tb/te... 09cf4f3c
/pxdefault_tb/te... a8f64332
/pxdefault_tb/te... 320b6a19
/pxdefault_tb/te... 01000100
/pxdefault_tb/te... 1
/pxdefault_tb/te... 0
/pxdefault_tb/te... 0
Now                 24438000 ps
Codelink Cursor     24438000 ps
```