ECE 385

Fall 2021

Experiment # 9

SOC with Advanced Encryption Standard in SystemVerilog

Xu Ke / Zhu Xiaohan

LA4/Thursday & 18:00-20:50 Huang Tianhao

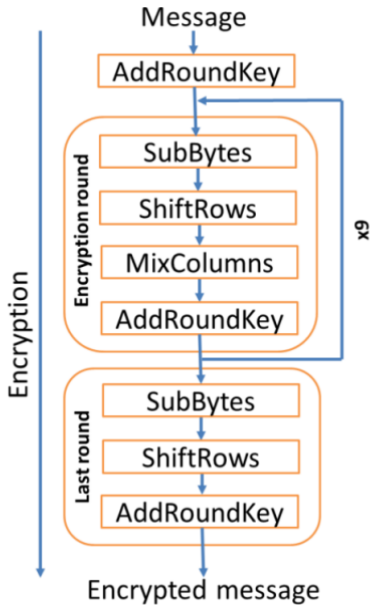
I. Introduction

In this experiment, we implement a 128-bit Advanced Encryption Standard (AES) using SystemVerilog as an Intellectual Property (IP) core. In the first part of this lab we will implement 128-bit AES encryption on the software IP core, and in the second part of this lab you will implement 128-bit AES decryption in SystemVerilog and design our own hardware IP core.

II. Written Description and Diagram of the AES encryptor/decryptor

1) Written description of the software encryptor

The NIOS-II core will perform as a platform for user interface and process the encryption function. It will receive users' input from the console. And print the encrypted result to the console and other information like the benchmark result. Additionally, it will also communicate with the hardware decryption core, and the result from it will also be sent to the NIOS-II core and printed at the console. For the software part of this lab, we write a AES encryption algorithm in the NIOS-II core. It basically has five different functions. The function flow is shown below:

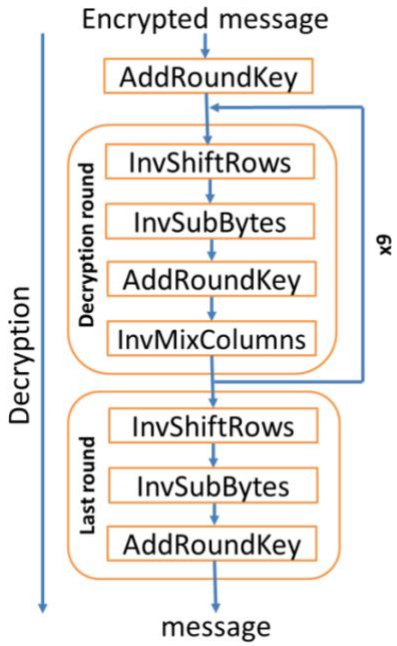


Firstly, it will perform KeyExpansion function. It will expand the round key based on the previous key values. These keys will be used in the AddRoundKey functions in each round of the program. The AddRoundKey is used to update the current state by XOR a 4-word key matrix generated by the key expansion function. In the SubBytes function, we have a S-Box storing 256 Bytes array. Each value in the state matrix will be substituted by the value in this S-Box based on the value of itself. The ShiftRows function will update the state by cyclically shifting with a certain offset. MixColumns function will perform a revised dot product by a fixed polynomial matrix c(x). Each column will be perform multiplication over GF(2^8).

```
AES(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
      byte state[4,Nb]
      state = in
      AddRoundKey(state, w[0, Nb-1])
      for round = 1 step 1 to Nr-1
          SubBytes(state)
          ShiftRows(state)
          MixColumns(state)
          AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      end for
      SubBytes(state)
      ShiftRows(state)
      AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
      out = state
end
```

2) Written description of the hardware decryptor

The process of decryption is similar to the encryption. The diagram of the AES decryption algorithm is shown below:



It has five functions: KeyExpansion, AddRoundKey, InvShiftRows, InvSubBytes, InvMixColumns. The KeyExpansion and AddRoundKey function is the same as the encryption algorithm. For InvShiftRows, it has the similar process as ShiftRow algorithm, the only difference is that it shifts rightwards. The InvSubBytes function is also similar as the SubBytes function in the encryption process, the difference is that it will use a different S-Box for substitution. The InvMixColumns is also similar to the MixColumns function in encryption process, the difference is that it will be multiplied with a different polynomial matrix c(x). And from the algorithm flow figure above, we can also see that the order of functions is also different to the encryption process. For the hardware performance, we use a

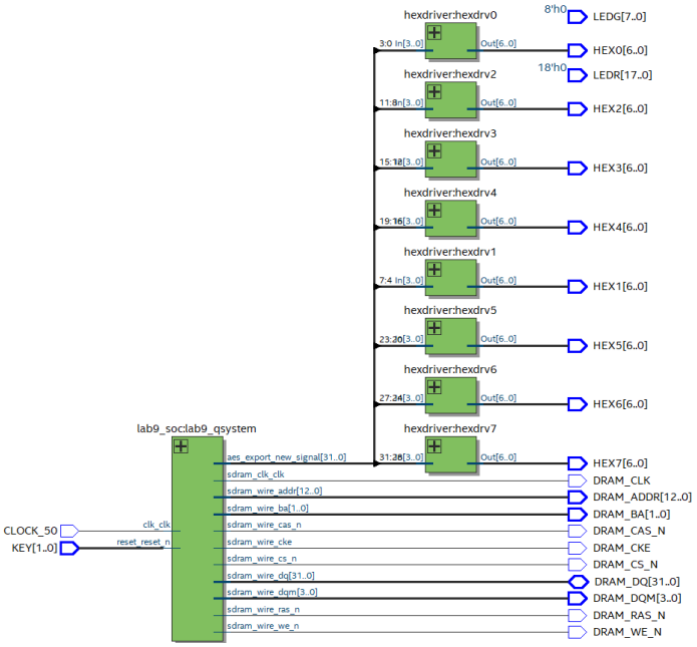
state machine to control the process of decryption. Each function is a circuit module used in the IP core, and it will be controlled through the state machine by the control signals input to them.

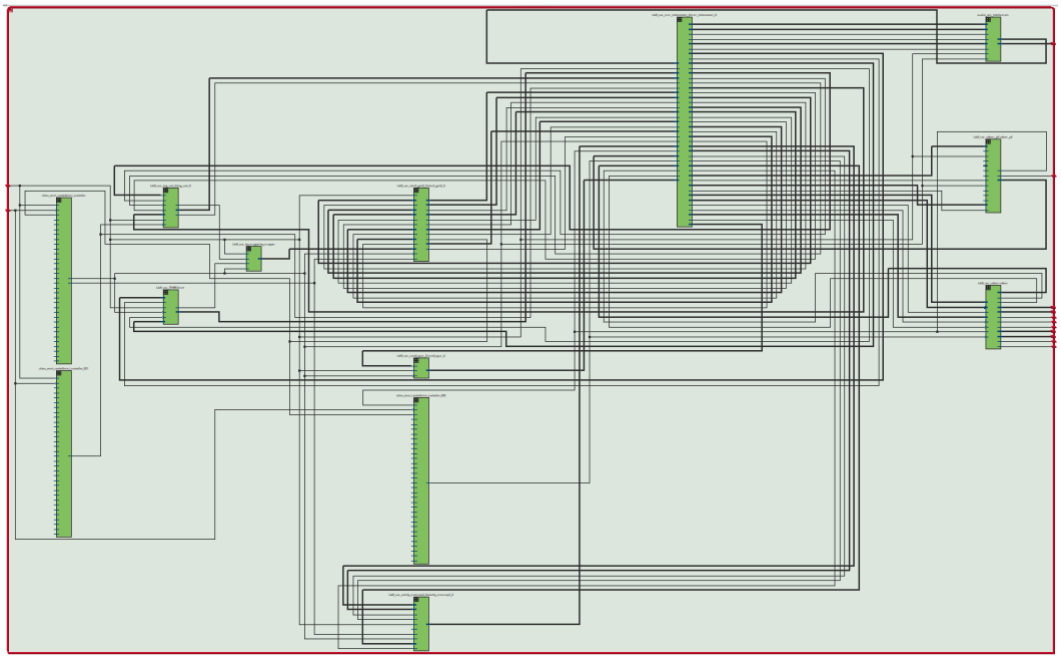
```
InvAES(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
      byte state[4,Nb]
      state = in
      AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
      for round = Nr-1 step -1 downto 1
          InvShiftRows(state)
          InvSubBytes(state)
          AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
          InvMixColumns(state)
      end for
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[0, Nb-1])
      out = state
end
```

3) Written description of the hardware/software interface

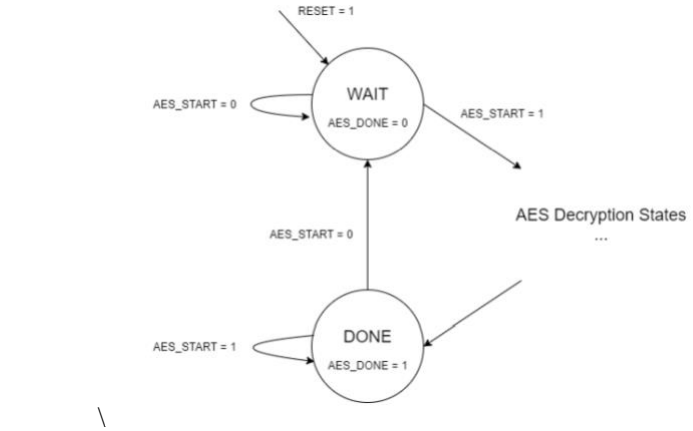
This module serves as the interface between the decryption IP core and the software encryption function. In this module, we have sixteen 32-bit registers to store the values for AES keys, encryption message and decryption message. All these register files will be connected to the Avalon-MM port to the NIOS-core to communicate with the program. And from the other side, these register files will also be connected to the AES module, which is the decryption core to communicate with it.

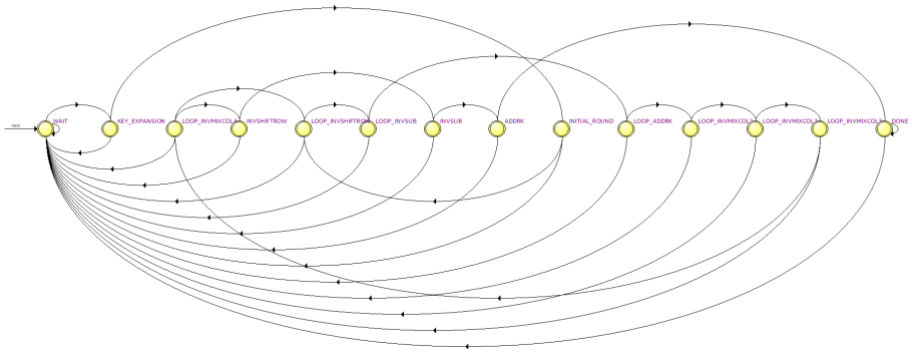
4) Block Diagram





5) State Diagram of AES Decryptor Controller





The states from left to right are WAIT, KEY_EXPANSION, LOOP_INVMIXCOL4, INVSHIFTROW, LOOP_INVSHIFTROW, LOOP_INVSUB, INVSUB, ADDK, INITIAL_ROUND, LOOP_ADDK, LOOP_INVMIXCOL1, LOOP_INVMIXCOL2, LOOP_INVMIXCOL3, DONE. The states between WAIT and DONE are used to control the decryption process. INITIAL_ROUND is for the first addroundkey.

LOOP_INVSHIFTROW, LOOP_INVSUB, LOOP_ADDK, LOOP_INVMIXCOL1, LOOP_INVMIXCOL2, LOOP_INVMIXCOL3, LOOP_INVMIXCOL4 are used for the 9 loops in the decryption round part. INVSHIFTROW, INVSUB, ADDK are used for the last round.

6) Module Descriptions

```
module lab9_top (
    input  logic        CLOCK_50,
    input  logic [1:0]  KEY,
    output logic [7:0]  LEDG,
    output logic [17:0] LEDR,
    output logic [6:0]  HEX0,
    output logic [6:0]  HEX1,
    output logic [6:0]  HEX2,
    output logic [6:0]  HEX3,
    output logic [6:0]  HEX4,
    output logic [6:0]  HEX5,
    output logic [6:0]  HEX6,
    output logic [6:0]  HEX7,
    output logic [12:0] DRAM_ADDR,
    output logic [1:0]  DRAM_BA,
    output logic        DRAM_CAS_N,
    output logic        DRAM_CKE,
    output logic        DRAM_CS_N,
    inout  logic [31:0] DRAM_DQ,
    output logic [3:0]  DRAM_DQM,
    output logic        DRAM_RAS_N,
    output logic        DRAM_WE_N,
    output logic        DRAM_CLK
);

// Exported data to show on Hex displays
logic [31:0] AES_EXPORT_DATA;

// Instantiation of Qsys design
lab9_soc lab9_qsystem (
    .clk_clk(CLOCK_50),                     // Clock input
    .reset_reset_n(KEY[0]),                 // Reset key
    .aes_export_new_signal(AES_EXPORT_DATA), // Exported data
    .sdram_wire_addr(DRAM_ADDR),            // sdram_wire.addr
    .sdram_wire_ba(DRAM_BA),                // sdram_wire.ba
    .sdram_wire_cas_n(DRAM_CAS_N),          // sdram_wire.cas_n
    .sdram_wire_cke(DRAM_CKE),              // sdram_wire.cke
    .sdram_wire_cs_n(DRAM_CS_N),            // sdram.cs_n
    .sdram_wire_dq(DRAM_DQ),                // sdram.dq
    .sdram_wire_dqm(DRAM_DQM),              // sdram.dqm
    .sdram_wire_ras_n(DRAM_RAS_N),          // sdram.ras_n
    .sdram_wire_we_n(DRAM_WE_N),            // sdram.we_n
    .sdram_clk_clk(DRAM_CLK)                // Clock out to SDRAM
);

// Display the first 4 and the last 4 hex values of the received message
hexdriver hexdrv0 (
    .In(AES_EXPORT_DATA[3:0]),
    .Out(HEX0)
);
hexdriver hexdrv1 (
    .In(AES_EXPORT_DATA[7:4]),
    .Out(HEX1)
);
hexdriver hexdrv2 (
    .In(AES_EXPORT_DATA[11:8]),
    .Out(HEX2)
);
hexdriver hexdrv3 (
    .In(AES_EXPORT_DATA[15:12]),
    .Out(HEX3)
);
hexdriver hexdrv4 (
   .In(AES_EXPORT_DATA[19:16]),
   .Out(HEX4)
);
hexdriver hexdrv5 (
   .In(AES_EXPORT_DATA[23:20]),
   .Out(HEX5)
);
hexdriver hexdrv6 (
   .In(AES_EXPORT_DATA[27:24]),
   .Out(HEX6)
);
hexdriver hexdrv7 (
   .In(AES_EXPORT_DATA[31:28]),
   .Out(HEX7)
);

endmodule
```

Module: lab9_top.sv

Inputs: CLOCK_50, [1:0] KEY

Outputs: DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK, [7:0] LEDG, [17:0] LEDR, [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7, [12:0] DRAM_ADDR, [1:0] DRAM_BA, [3:0] DRAM_DQM

Inout: [31:0] DRAM_DQ

Description: This is the top-level file, which instantiates lab9_soc from the qsys files, provides data and displays values through hex drivers.

Purpose: Give the top-level structure of the system.

```
module SubBytes (
    input  logic clk,
    input  logic [7:0] in,
    output logic [7:0] out
);

// This module will be synthesized into a RAM
always_ff @ (negedge clk)
    case (in)
    8'h00: out <= 8'h63;    8'h01: out <= 8'h7c;    8'h02: out <= 8'h77;    8'h03: out <= 8'h7b;
    8'h04: out <= 8'hf2;    8'h05: out <= 8'h6b;    8'h06: out <= 8'h6f;    8'h07: out <= 8'hc5;
    8'h08: out <= 8'h30;    8'h09: out <= 8'h01;    8'h0a: out <= 8'h67;    8'h0b: out <= 8'h2b;
    8'h0c: out <= 8'hfe;    8'h0d: out <= 8'hd7;    8'h0e: out <= 8'hab;    8'h0f: out <= 8'h76;
    8'h10: out <= 8'hca;    8'h11: out <= 8'h82;    8'h12: out <= 8'hc9;    8'h13: out <= 8'h7d;
    8'h14: out <= 8'hfa;    8'h15: out <= 8'h59;    8'h16: out <= 8'h47;    8'h17: out <= 8'hf0;
    8'h18: out <= 8'had;    8'h19: out <= 8'hd4;    8'h1a: out <= 8'ha2;    8'h1b: out <= 8'haf;
    8'h1c: out <= 8'h9c;    8'h1d: out <= 8'ha4;    8'h1e: out <= 8'h72;    8'h1f: out <= 8'hc0;
    8'h20: out <= 8'hb7;    8'h21: out <= 8'hfd;    8'h22: out <= 8'h93;    8'h23: out <= 8'h26;
    8'h24: out <= 8'h36;    8'h25: out <= 8'h3f;    8'h26: out <= 8'hf7;    8'h27: out <= 8'hcc;
    8'h28: out <= 8'h34;    8'h29: out <= 8'ha5;    8'h2a: out <= 8'he5;    8'h2b: out <= 8'hf1;
    8'h2c: out <= 8'h71;    8'h2d: out <= 8'hd8;    8'h2e: out <= 8'h31;    8'h2f: out <= 8'h15;
    8'h30: out <= 8'h04;    8'h31: out <= 8'hc7;    8'h32: out <= 8'h23;    8'h33: out <= 8'hc3;
    8'h34: out <= 8'h18;    8'h35: out <= 8'h96;    8'h36: out <= 8'h05;    8'h37: out <= 8'h9a;
    8'h38: out <= 8'h07;    8'h39: out <= 8'h12;    8'h3a: out <= 8'h80;    8'h3b: out <= 8'he2;
    8'h3c: out <= 8'heb;    8'h3d: out <= 8'h27;    8'h3e: out <= 8'hb2;    8'h3f: out <= 8'h75;
    8'h40: out <= 8'h09;    8'h41: out <= 8'h83;    8'h42: out <= 8'h2c;    8'h43: out <= 8'h1a;
    8'h44: out <= 8'h1b;    8'h45: out <= 8'h6e;    8'h46: out <= 8'h5a;    8'h47: out <= 8'ha0;
    8'h48: out <= 8'h52;    8'h49: out <= 8'h3b;    8'h4a: out <= 8'hd6;    8'h4b: out <= 8'hb3;
    8'h4c: out <= 8'h29;    8'h4d: out <= 8'he3;    8'h4e: out <= 8'h2f;    8'h4f: out <= 8'h84;
    8'h50: out <= 8'h53;    8'h51: out <= 8'hd1;    8'h52: out <= 8'h00;    8'h53: out <= 8'hed;
    8'h54: out <= 8'h20;    8'h55: out <= 8'hfc;    8'h56: out <= 8'hb1;    8'h57: out <= 8'h5b;
    8'h58: out <= 8'h6a;    8'h59: out <= 8'hcb;    8'h5a: out <= 8'hbe;    8'h5b: out <= 8'h39;
    8'h5c: out <= 8'h4a;    8'h5d: out <= 8'h4c;    8'h5e: out <= 8'h58;    8'h5f: out <= 8'hcf;
    8'h60: out <= 8'hd0;    8'h61: out <= 8'hef;    8'h62: out <= 8'haa;    8'h63: out <= 8'hfb;
    8'h64: out <= 8'h43;    8'h65: out <= 8'h4d;    8'h66: out <= 8'h33;    8'h67: out <= 8'h85;
    8'h68: out <= 8'h45;    8'h69: out <= 8'hf9;    8'h6a: out <= 8'h02;    8'h6b: out <= 8'h7f;
    8'h6c: out <= 8'h50;    8'h6d: out <= 8'h3c;    8'h6e: out <= 8'h9f;    8'h6f: out <= 8'ha8;
    8'h70: out <= 8'h51;    8'h71: out <= 8'ha3;    8'h72: out <= 8'h40;    8'h73: out <= 8'h8f;
    8'h74: out <= 8'h92;    8'h75: out <= 8'h9d;    8'h76: out <= 8'h38;    8'h77: out <= 8'hf5;
    8'h78: out <= 8'hbc;    8'h79: out <= 8'hb6;    8'h7a: out <= 8'hda;    8'h7b: out <= 8'h21;
    8'h7c: out <= 8'h10;    8'h7d: out <= 8'hff;    8'h7e: out <= 8'hf3;    8'h7f: out <= 8'hd2;
    8'h80: out <= 8'hcd;    8'h81: out <= 8'h0c;    8'h82: out <= 8'h13;    8'h83: out <= 8'hec;
    8'h84: out <= 8'h5f;    8'h85: out <= 8'h97;    8'h86: out <= 8'h44;    8'h87: out <= 8'h17;
    8'h88: out <= 8'hc4;    8'h89: out <= 8'ha7;    8'h8a: out <= 8'h7e;    8'h8b: out <= 8'h3d;
    8'h8c: out <= 8'h64;    8'h8d: out <= 8'h5d;    8'h8e: out <= 8'h19;    8'h8f: out <= 8'h73;
    8'h90: out <= 8'h60;    8'h91: out <= 8'h81;    8'h92: out <= 8'h4f;    8'h93: out <= 8'hdc;
    8'h94: out <= 8'h22;    8'h95: out <= 8'h2a;    8'h96: out <= 8'h90;    8'h97: out <= 8'h88;
    8'h98: out <= 8'h46;    8'h99: out <= 8'hee;    8'h9a: out <= 8'hb8;    8'h9b: out <= 8'h14;
    8'h9c: out <= 8'hde;    8'h9d: out <= 8'h5e;    8'h9e: out <= 8'h0b;    8'h9f: out <= 8'hdb;
    8'ha0: out <= 8'he0;    8'ha1: out <= 8'h32;    8'ha2: out <= 8'h3a;    8'ha3: out <= 8'h0a;
    8'ha4: out <= 8'h49;    8'ha5: out <= 8'h06;    8'ha6: out <= 8'h24;    8'ha7: out <= 8'h5c;
    8'ha8: out <= 8'hc2;    8'ha9: out <= 8'hd3;    8'haa: out <= 8'hac;    8'hab: out <= 8'h62;
    8'hac: out <= 8'h91;    8'had: out <= 8'h95;    8'hae: out <= 8'he4;    8'haf: out <= 8'h79;
    8'hb0: out <= 8'he7;    8'hb1: out <= 8'hc8;    8'hb2: out <= 8'h37;    8'hb3: out <= 8'h6d;
    8'hb4: out <= 8'h8d;    8'hb5: out <= 8'hd5;    8'hb6: out <= 8'h4e;    8'hb7: out <= 8'ha9;
    8'hb8: out <= 8'h6c;    8'hb9: out <= 8'h56;    8'hba: out <= 8'hf4;    8'hbb: out <= 8'hea;
    8'hbc: out <= 8'h65;    8'hbd: out <= 8'h7a;    8'hbe: out <= 8'hae;    8'hbf: out <= 8'h08;
    8'hc0: out <= 8'hba;    8'hc1: out <= 8'h78;    8'hc2: out <= 8'h25;    8'hc3: out <= 8'h2e;
    8'hc4: out <= 8'h1c;    8'hc5: out <= 8'ha6;    8'hc6: out <= 8'hb4;    8'hc7: out <= 8'hc6;
    8'hc8: out <= 8'he8;    8'hc9: out <= 8'hdd;    8'hca: out <= 8'h74;    8'hcb: out <= 8'h1f;
    8'hcc: out <= 8'h4b;    8'hcd: out <= 8'hbd;    8'hce: out <= 8'h8b;    8'hcf: out <= 8'h8a;
    8'hd0: out <= 8'h70;    8'hd1: out <= 8'h3e;    8'hd2: out <= 8'hb5;    8'hd3: out <= 8'h66;
    8'hd4: out <= 8'h48;    8'hd5: out <= 8'h03;    8'hd6: out <= 8'hf6;    8'hd7: out <= 8'h0e;
    8'hd8: out <= 8'h61;    8'hd9: out <= 8'h35;    8'hda: out <= 8'h57;    8'hdb: out <= 8'hb9;
    8'hdc: out <= 8'h86;    8'hdd: out <= 8'hc1;    8'hde: out <= 8'h1d;    8'hdf: out <= 8'h9e;
    8'he0: out <= 8'he1;    8'he1: out <= 8'hf8;    8'he2: out <= 8'h98;    8'he3: out <= 8'h11;
    8'he4: out <= 8'h69;    8'he5: out <= 8'hd9;    8'he6: out <= 8'h8e;    8'he7: out <= 8'h94;
    8'he8: out <= 8'h9b;    8'he9: out <= 8'h1e;    8'hea: out <= 8'h87;    8'heb: out <= 8'he9;
    8'hec: out <= 8'hce;    8'hed: out <= 8'h55;    8'hee: out <= 8'h28;    8'hef: out <= 8'hdf;
    8'hf0: out <= 8'h8c;    8'hf1: out <= 8'ha1;    8'hf2: out <= 8'h89;    8'hf3: out <= 8'h0d;
    8'hf4: out <= 8'hbf;    8'hf5: out <= 8'he6;    8'hf6: out <= 8'h42;    8'hf7: out <= 8'h68;
    8'hf8: out <= 8'h41;    8'hf9: out <= 8'h99;    8'hfa: out <= 8'h2d;    8'hfb: out <= 8'h0f;
    8'hfc: out <= 8'hb0;    8'hfd: out <= 8'h54;    8'hfe: out <= 8'hbb;    8'hff: out <= 8'h16;
    endcase
endmodule
```

Module: SubBytes

Inputs: clk, [7:0] in

Output: [7:0] out

Description: This module takes a byte as the state and lookup in the table stored in it and give an encrypted byte of the corresponding state.

Purpose: It serves as the SubBytes function in the encryption process and would serve the decryption process if it was implemented within the hardware.

```
module InvSubBytes (
   input  logic clk,
   input  logic [7:0] in,
   output logic [7:0] out
);

// This module will be synthesized into a RAM
always_ff @ (negedge clk)
   case (in)
   8'h00: out <= 8'h52;    8'h01: out <= 8'h09;    8'h02: out <= 8'h6a;    8'h03: out <= 8'hd5;
   8'h04: out <= 8'h30;    8'h05: out <= 8'h36;    8'h06: out <= 8'ha5;    8'h07: out <= 8'h38;
   8'h08: out <= 8'hbf;    8'h09: out <= 8'h40;    8'h0a: out <= 8'ha3;    8'h0b: out <= 8'h9e;
   8'h0c: out <= 8'h81;    8'h0d: out <= 8'hf3;    8'h0e: out <= 8'hd7;    8'h0f: out <= 8'hfb;
   8'h10: out <= 8'h7c;    8'h11: out <= 8'he3;    8'h12: out <= 8'h39;    8'h13: out <= 8'h82;
   8'h14: out <= 8'h9b;    8'h15: out <= 8'h2f;    8'h16: out <= 8'hff;    8'h17: out <= 8'h87;
   8'h18: out <= 8'h34;    8'h19: out <= 8'h8e;    8'h1a: out <= 8'h43;    8'h1b: out <= 8'h44;
   8'h1c: out <= 8'hc4;    8'h1d: out <= 8'hde;    8'h1e: out <= 8'he9;    8'h1f: out <= 8'hcb;
   8'h20: out <= 8'h54;    8'h21: out <= 8'h7b;    8'h22: out <= 8'h94;    8'h23: out <= 8'h32;
   8'h24: out <= 8'ha6;    8'h25: out <= 8'hc2;    8'h26: out <= 8'h23;    8'h27: out <= 8'h3d;
   8'h28: out <= 8'hee;    8'h29: out <= 8'h4c;    8'h2a: out <= 8'h95;    8'h2b: out <= 8'h0b;
   8'h2c: out <= 8'h42;    8'h2d: out <= 8'hfa;    8'h2e: out <= 8'hc3;    8'h2f: out <= 8'h4e;
   8'h30: out <= 8'h08;    8'h31: out <= 8'h2e;    8'h32: out <= 8'ha1;    8'h33: out <= 8'h66;
   8'h34: out <= 8'h28;    8'h35: out <= 8'hd9;    8'h36: out <= 8'h24;    8'h37: out <= 8'hb2;
   8'h38: out <= 8'h76;    8'h39: out <= 8'h5b;    8'h3a: out <= 8'ha2;    8'h3b: out <= 8'h49;
   8'h3c: out <= 8'h6d;    8'h3d: out <= 8'h8b;    8'h3e: out <= 8'hd1;    8'h3f: out <= 8'h25;
   8'h40: out <= 8'h72;    8'h41: out <= 8'hf8;    8'h42: out <= 8'hf6;    8'h43: out <= 8'h64;
   8'h44: out <= 8'h86;    8'h45: out <= 8'h68;    8'h46: out <= 8'h98;    8'h47: out <= 8'h16;
   8'h48: out <= 8'hd4;    8'h49: out <= 8'ha4;    8'h4a: out <= 8'h5c;    8'h4b: out <= 8'hcc;
   8'h4c: out <= 8'h5d;    8'h4d: out <= 8'h65;    8'h4e: out <= 8'hb6;    8'h4f: out <= 8'h92;
   8'h50: out <= 8'h6c;    8'h51: out <= 8'h70;    8'h52: out <= 8'h48;    8'h53: out <= 8'h50;
   8'h54: out <= 8'hfd;    8'h55: out <= 8'hed;    8'h56: out <= 8'hb9;    8'h57: out <= 8'hda;
   8'h58: out <= 8'h5e;    8'h59: out <= 8'h15;    8'h5a: out <= 8'h46;    8'h5b: out <= 8'h57;
   8'h5c: out <= 8'ha7;    8'h5d: out <= 8'h8d;    8'h5e: out <= 8'h9d;    8'h5f: out <= 8'h84;
   8'h60: out <= 8'h90;    8'h61: out <= 8'hd8;    8'h62: out <= 8'hab;    8'h63: out <= 8'h00;
   8'h64: out <= 8'h8c;    8'h65: out <= 8'hbc;    8'h66: out <= 8'hd3;    8'h67: out <= 8'h0a;
   8'h68: out <= 8'hf7;    8'h69: out <= 8'he4;    8'h6a: out <= 8'h58;    8'h6b: out <= 8'h05;
   8'h6c: out <= 8'hb8;    8'h6d: out <= 8'hb3;    8'h6e: out <= 8'h45;    8'h6f: out <= 8'h06;
   8'h70: out <= 8'hd0;    8'h71: out <= 8'h2c;    8'h72: out <= 8'h1e;    8'h73: out <= 8'h8f;
   8'h74: out <= 8'hca;    8'h75: out <= 8'h3f;    8'h76: out <= 8'h0f;    8'h77: out <= 8'h02;
   8'h78: out <= 8'hc1;    8'h79: out <= 8'haf;    8'h7a: out <= 8'hbd;    8'h7b: out <= 8'h03;
   8'h7c: out <= 8'h01;    8'h7d: out <= 8'h13;    8'h7e: out <= 8'h8a;    8'h7f: out <= 8'h6b;
   8'h80: out <= 8'h3a;    8'h81: out <= 8'h91;    8'h82: out <= 8'h11;    8'h83: out <= 8'h41;
   8'h84: out <= 8'h4f;    8'h85: out <= 8'h67;    8'h86: out <= 8'hdc;    8'h87: out <= 8'hea;
   8'h88: out <= 8'h97;    8'h89: out <= 8'hf2;    8'h8a: out <= 8'hcf;    8'h8b: out <= 8'hce;
   8'h8c: out <= 8'hf0;    8'h8d: out <= 8'hb4;    8'h8e: out <= 8'he6;    8'h8f: out <= 8'h73;
   8'h90: out <= 8'h96;    8'h91: out <= 8'hac;    8'h92: out <= 8'h74;    8'h93: out <= 8'h22;
   8'h94: out <= 8'he7;    8'h95: out <= 8'had;    8'h96: out <= 8'h35;    8'h97: out <= 8'h85;
   8'h98: out <= 8'he2;    8'h99: out <= 8'hf9;    8'h9a: out <= 8'h37;    8'h9b: out <= 8'he8;
   8'h9c: out <= 8'h1c;    8'h9d: out <= 8'h75;    8'h9e: out <= 8'hdf;    8'h9f: out <= 8'h6e;
   8'ha0: out <= 8'h47;    8'ha1: out <= 8'hf1;    8'ha2: out <= 8'h1a;    8'ha3: out <= 8'h71;
   8'ha4: out <= 8'h1d;    8'ha5: out <= 8'h29;    8'ha6: out <= 8'hc5;    8'ha7: out <= 8'h89;
   8'ha8: out <= 8'h6f;    8'ha9: out <= 8'hb7;    8'haa: out <= 8'h62;    8'hab: out <= 8'h0e;
   8'hac: out <= 8'haa;    8'had: out <= 8'h18;    8'hae: out <= 8'hbe;    8'haf: out <= 8'h1b;
   8'hb0: out <= 8'hfc;    8'hb1: out <= 8'h56;    8'hb2: out <= 8'h3e;    8'hb3: out <= 8'h4b;
   8'hb4: out <= 8'hc6;    8'hb5: out <= 8'hd2;    8'hb6: out <= 8'h79;    8'hb7: out <= 8'h20;
   8'hb8: out <= 8'h9a;    8'hb9: out <= 8'hdb;    8'hba: out <= 8'hc0;    8'hbb: out <= 8'hfe;
   8'hbc: out <= 8'h78;    8'hbd: out <= 8'hcd;    8'hbe: out <= 8'h5a;    8'hbf: out <= 8'hf4;
   8'hc0: out <= 8'h1f;    8'hc1: out <= 8'hdd;    8'hc2: out <= 8'ha8;    8'hc3: out <= 8'h33;
   8'hc4: out <= 8'h88;    8'hc5: out <= 8'h07;    8'hc6: out <= 8'hc7;    8'hc7: out <= 8'h31;
   8'hc8: out <= 8'hb1;    8'hc9: out <= 8'h12;    8'hca: out <= 8'h10;    8'hcb: out <= 8'h59;
   8'hcc: out <= 8'h27;    8'hcd: out <= 8'h80;    8'hce: out <= 8'hec;    8'hcf: out <= 8'h5f;
   8'hd0: out <= 8'h60;    8'hd1: out <= 8'h51;    8'hd2: out <= 8'h7f;    8'hd3: out <= 8'ha9;
   8'hd4: out <= 8'h19;    8'hd5: out <= 8'hb5;    8'hd6: out <= 8'h4a;    8'hd7: out <= 8'h0d;
   8'hd8: out <= 8'h2d;    8'hd9: out <= 8'he5;    8'hda: out <= 8'h7a;    8'hdb: out <= 8'h9f;
   8'hdc: out <= 8'h93;    8'hdd: out <= 8'hc9;    8'hde: out <= 8'h9c;    8'hdf: out <= 8'hef;
   8'he0: out <= 8'ha0;    8'he1: out <= 8'he0;    8'he2: out <= 8'h3b;    8'he3: out <= 8'h4d;
   8'he4: out <= 8'hae;    8'he5: out <= 8'h2a;    8'he6: out <= 8'hf5;    8'he7: out <= 8'hb0;
   8'he8: out <= 8'hc8;    8'he9: out <= 8'heb;    8'hea: out <= 8'hbb;    8'heb: out <= 8'h3c;
   8'hec: out <= 8'h83;    8'hed: out <= 8'h53;    8'hee: out <= 8'h99;    8'hef: out <= 8'h61;
   8'hf0: out <= 8'h17;    8'hf1: out <= 8'h2b;    8'hf2: out <= 8'h04;    8'hf3: out <= 8'h7e;
   8'hf4: out <= 8'hba;    8'hf5: out <= 8'h77;    8'hf6: out <= 8'hd6;    8'hf7: out <= 8'h26;
   8'hf8: out <= 8'he1;    8'hf9: out <= 8'h69;    8'hfa: out <= 8'h14;    8'hfb: out <= 8'h63;
   8'hfc: out <= 8'h55;    8'hfd: out <= 8'h21;    8'hfe: out <= 8'h0c;    8'hff: out <= 8'h7d;
   endcase
endmodule
```

Module: InvSubBytes

Inputs: clk, [7:0] in

Output: [7:0] out

Description: This module takes a byte as the state and lookup in the table stored in it and give a decrypted byte of the corresponding state.

Purpose: It serves as the InvSubBytes function in the decryption process.

```
// Note that KeyExpansion does not complete in single clock cycle.
// Run simulation to see how many clock cycles it takes for key schedule to complete.
module KeyExpansion (
   input  logic clk,
   input  logic [127:0]  Cipherkey,
   output logic [1407:0] KeySchedule
);

// Rcon table
const byte unsigned Rcon [0:9] = '{ 8'h01,8'h02,8'h04,8'h08,8'h10,
                                    8'h20,8'h40,8'h80,8'h1b,8'h36 };

logic [127:0] key0, key1, key2, key3, key4, key5, key6, key7, key8, key9;

KeyExpansionOne key_0 (.*, .oldkey(Cipherkey), .newkey(key0), .Rcon(Rcon[0]));
KeyExpansionOne key_1 (.*, .oldkey(key0), .newkey(key1), .Rcon(Rcon[1]));
KeyExpansionOne key_2 (.*, .oldkey(key1), .newkey(key2), .Rcon(Rcon[2]));
KeyExpansionOne key_3 (.*, .oldkey(key2), .newkey(key3), .Rcon(Rcon[3]));
KeyExpansionOne key_4 (.*, .oldkey(key3), .newkey(key4), .Rcon(Rcon[4]));
KeyExpansionOne key_5 (.*, .oldkey(key4), .newkey(key5), .Rcon(Rcon[5]));
KeyExpansionOne key_6 (.*, .oldkey(key5), .newkey(key6), .Rcon(Rcon[6]));
KeyExpansionOne key_7 (.*, .oldkey(key6), .newkey(key7), .Rcon(Rcon[7]));
KeyExpansionOne key_8 (.*, .oldkey(key7), .newkey(key8), .Rcon(Rcon[8]));
KeyExpansionOne key_9 (.*, .oldkey(key8), .newkey(key9), .Rcon(Rcon[9]));

assign KeySchedule[1407:0] = {Cipherkey, key0, key1, key2, key3, key4, key5, key6, key7, key8, key9};

endmodule
```

Module: KeyExpansion

Inputs: clk, [127:0] Cipherkey

Output: [1407:0] KeySchedule

Description: This module instantiates the KeyExpansionOne module for 10 times and generate a keyschedule for the decryption process.

Purpose: This module serves as the KeyExpansion function in the decryption process to provide a key schedule for the decryption.

```
module KeyExpansionOne (
   input  logic clk,
   input  logic [127:0] oldkey,
   output logic [127:0] newkey,
   input  logic [7:0] Rcon
);

   // Obtain words from previous key
   logic [31:0] wp0, wp1, wp2, wp3;
   logic [7:0]  wp03, wp13, wp23, wp33;
   logic [31:0] subword;
   assign {wp0, wp1, wp2, wp3} = oldkey;
   assign {wp03, wp13, wp23, wp33} = wp3;

   // Obtain SubWord using SubBytes.  Notice that RotWord is implemented using rotated input
   SubBytes subbytes_0(.*, .in(wp13), .out(subword[31:24]));
   SubBytes subbytes_1(.*, .in(wp23), .out(subword[23:16]));
   SubBytes subbytes_2(.*, .in(wp33), .out(subword[15:8]));
   SubBytes subbytes_3(.*, .in(wp03), .out(subword[7:0]));

   logic [31:0] w0, w1, w2, w3;
   assign w0 = wp0 ^ {subword[31:24] ^ Rcon, subword[23:0]};
   assign w1 = wp1 ^ w0;
   assign w2 = wp2 ^ w1;
   assign w3 = wp3 ^ w2;

   assign newkey = {w0, w1, w2, w3};

endmodule
```

Module: KeyExpansionOne

Inputs: clk, [127:0] oldkey, [7:0] Rcon

Output: [127:0] newkey

Description: It takes Rcon and a given bar of key, by XOR operation, it generates a bar of keyschedule.

Purpose: It’s the component of KeyExpansion to calculate a unit bar of keyschedule.

```
module InvShiftRows (input logic [127:0] data_in, output logic [127:0] data_out);

logic [0:127] in;
logic [0:127] out;

assign in = data_in;
assign data_out = out;

logic [7:0] a0 [0:3];
logic [7:0] a1 [0:3];
logic [7:0] a2 [0:3];
logic [7:0] a3 [0:3];

assign a0[0] = in[0:7];
assign a1[0] = in[8:15];
assign a2[0] = in[16:23];
assign a3[0] = in[24:31];

assign a0[1] = in[32:39];
assign a1[1] = in[40:47];
assign a2[1] = in[48:55];
assign a3[1] = in[56:63];

assign a0[2] = in[64:71];
assign a1[2] = in[72:79];
assign a2[2] = in[80:87];
assign a3[2] = in[88:95];

assign a0[3] = in[96:103];
assign a1[3] = in[104:111];
assign a2[3] = in[112:119];
assign a3[3] = in[120:127];

assign out[0:7] = a0[0];
assign out[8:15] = a1[3];
assign out[16:23] = a2[2];
assign out[24:31] = a3[1];

assign out[32:39] = a0[1];
assign out[40:47] = a1[0];
assign out[48:55] = a2[3];
assign out[56:63] = a3[2];

assign out[64:71] = a0[2];
assign out[72:79] = a1[1];
assign out[80:87] = a2[0];
assign out[88:95] = a3[3];

assign out[96:103] = a0[3];
assign out[104:111] = a1[2];
assign out[112:119] = a2[1];
assign out[120:127] = a3[0];

endmodule
```

Module: InvShiftRows.sv

Input: [127:0] data_in

Output: [127:0] data_out

Description: Consider the input data as a 4 by 4 matrix, it shifts rightwards each row. Specifically, row n is right-circularly shifted by n −1 Bytes.

Purpose: This module serves as the InvShiftRows function in the decryption process to shift rows of the aimed matrix.

```
module InvMixColumns (
    input  logic [31:0] in,
    output logic [31:0] out
);

// Declaration of the individual Bytes in Word
logic [7:0] a0, a1, a2, a3;
// Declaration of the finite field multiplication results
logic [7:0] b00, b01, b02, b03,
            b10, b11, b12, b13,
            b20, b21, b22, b23,
            b30, b31, b32, b33;

// Decompose the Word into 4 Bytes
assign {a0, a1, a2, a3} = in;

// Finite field multiplications with {09}
GF_Mul_9 gfmul9_0(a3, b03);
GF_Mul_9 gfmul9_1(a0, b10);
GF_Mul_9 gfmul9_2(a1, b21);
GF_Mul_9 gfmul9_3(a2, b32);

// Finite field multiplications with {0b}
GF_Mul_b gfmulb_0(a1, b01);
GF_Mul_b gfmulb_1(a2, b12);
GF_Mul_b gfmulb_2(a3, b23);
GF_Mul_b gfmulb_3(a0, b30);

// Finite field multiplications with {0d}
GF_Mul_d gfmuld_0(a2, b02);
GF_Mul_d gfmuld_1(a3, b13);
GF_Mul_d gfmuld_2(a0, b20);
GF_Mul_d gfmuld_3(a1, b31);

// Finite field multiplications with {0e}
GF_Mul_e gfmule_0(a0, b00);
GF_Mul_e gfmule_1(a1, b11);
GF_Mul_e gfmule_2(a2, b22);
GF_Mul_e gfmule_3(a3, b33);

// Assign output by XORing the above results
assign out[31:24] = b00^b01^b02^b03;
assign out[23:16] = b10^b11^b12^b13;
assign out[15: 8] = b20^b21^b22^b23;
assign out[ 7: 0] = b30^b31^b32^b33;

endmodule
```

Module: InvMixColumns.sv

Input: [31:0] in

Output: [31:0] out

Description: It takes a 32-bit word to perform the inverse MixColumns process in the decryption process.

Purpose: This module serves as the InvMixColumns function in the decryption process.

```
module hexdriver (
    input  logic [3:0] In,
    output logic [6:0] Out
);

always_comb begin
    unique case (In)
        4'b0000   : Out = 7'b1000000; // '0'
        4'b0001   : Out = 7'b1111001; // '1'
        4'b0010   : Out = 7'b0100100; // '2'
        4'b0011   : Out = 7'b0110000; // '3'
        4'b0100   : Out = 7'b0011001; // '4'
        4'b0101   : Out = 7'b0010010; // '5'
        4'b0110   : Out = 7'b0000010; // '6'
        4'b0111   : Out = 7'b1111000; // '7'
        4'b1000   : Out = 7'b0000000; // '8'
        4'b1001   : Out = 7'b0010000; // '9'
        4'b1010   : Out = 7'b0001000; // 'A'
        4'b1011   : Out = 7'b0000011; // 'b'
        4'b1100   : Out = 7'b1000110; // 'C'
        4'b1101   : Out = 7'b0100001; // 'd'
        4'b1110   : Out = 7'b0000110; // 'E'
        4'b1111   : Out = 7'b0001110; // 'F'
        default   : Out = 7'bX;
    endcase
end

endmodule
```

Module: hexdriver.sv

Input: [3:0] In

Output: [6:0] Out

Description: It maps the inputs to the seven-segment display module.

Purpose: It serves to display the aimed encryption and decryption information on FPGA.

```
module avalon_aes_interface (
   // Avalon Clock Input
   input logic CLK,

   // Avalon Reset Input
   input logic RESET,

   // Avalon-MM Slave Signals
   input  logic AVL_READ,                  // Avalon-MM Read
   input  logic AVL_WRITE,                 // Avalon-MM Write
   input  logic AVL_CS,                    // Avalon-MM Chip Select
   input  logic [3:0] AVL_BYTE_EN,         // Avalon-MM Byte Enable
   input  logic [3:0] AVL_ADDR,            // Avalon-MM Address
   input  logic [31:0] AVL_WRITEDATA,      // Avalon-MM Write Data
   output logic [31:0] AVL_READDATA,       // Avalon-MM Read Data

   // Exported Conduit
   output logic [31:0] EXPORT_DATA      // Exported Conduit Signal to LEDs
);
   logic [15:0][31:0] regfile;
   logic [3:0][31:0] MSG_DEC;
   logic Done;
   logic [127:0]next_state_out;


   integer i;

   always_ff @ (posedge CLK)
   begin
      if(RESET)
      begin
         for(i=0;i<16;i=i+1)
         begin
            regfile[i]<=32'b0;
         end
      end
      else if(AVL_WRITE && AVL_CS)
      begin
         case(AVL_BYTE_EN)
            4'b1111: regfile[AVL_ADDR] <= AVL_WRITEDATA;
            4'b1100: regfile[AVL_ADDR][31:16]<= AVL_WRITEDATA[31:16];
            4'b0011: regfile[AVL_ADDR][15:0] <= AVL_WRITEDATA[15:0];
            4'b1000: regfile[AVL_ADDR][31:24] <= AVL_WRITEDATA[31:24];
            4'b0100: regfile[AVL_ADDR][23:16] <= AVL_WRITEDATA[23:16];
            4'b0010: regfile[AVL_ADDR][15:8] <= AVL_WRITEDATA[15:8];
            4'b0001: regfile[AVL_ADDR][7:0] <= AVL_WRITEDATA[7:0];
         endcase
      end

//    else if(AVL_CS && regfile[15][0])
//    begin
////       regfile[15][0] = Done;
//       regfile[11:8] = MSG_DEC;
//    end

      else if(Done && AVL_CS)
      begin
         regfile[11:8] <= MSG_DEC;
         regfile[15][0] <= Done;
      end

//    else if(~Done && AVL_CS)
//    begin
//       regfile[11:8] <= next_state_out;
//    end


   end
```

```
always_comb
begin
   EXPORT_DATA = {regfile[3][31:16], regfile[0][15:0]};

   if(AVL_READ)
      AVL_READDATA = regfile[AVL_ADDR];
   else
      AVL_READDATA = 32'b0;
end

AES aes(
.CLK(CLK),
.RESET(RESET),
.AES_START(regfile[14][0]),
.AES_DONE(Done),
.AES_KEY(regfile[3:0]),
.AES_MSG_ENC(regfile[7:4]),
.AES_MSG_DEC(MSG_DEC),
.next_state_out(next_state_out));


endmodule
```

Module: avalon_aes_interface.sv

Inputs: CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS, [3:0] AVL_BYTE_EN, AVL_ADDR, [31:0] AVL_WRITEDATA

Outputs: [31:0] AVL_READDATA, EXPORT_DATA

Description: This module contains the register files, logic for register R/W in the bank. It serves as the interface of hardware and the SystemVerilog program and the instantiation of AES.sv.

Purpose: It serves as the primary access of SystemVerilog and the interface with AES_Decryption_Core in qsys, organizes the data for each hardware, software and SystemVerilog.

```
module AES (
    input  logic CLK,
    input  logic RESET,
    input  logic AES_START,
    output logic AES_DONE,
    input  logic [127:0] AES_KEY,
    input  logic [127:0] AES_MSG_ENC,
    output logic [127:0] AES_MSG_DEC,
    output logic [127:0] next_state_out
);

    logic [1407:0] KeySchedule;
    logic [127:0] state;
    logic [127:0] next_state;
    logic [127:0] key;
    logic [127:0] addroundkey_out;
    logic [127:0] invshiftrows_out;
    logic [31:0] mixcolumns_in;
    logic [31:0] mixcolumns_out;
    logic [127:0] Sub_Out;

    enum logic [4:0]{
        WAIT,
        DONE,
        KEY_EXPANSION,
        INITIAL_ROUND,

        LOOP_INVSUB,
        LOOP_INVSHIFTROW,
        LOOP_INVMIXCOL1,
        LOOP_INVMIXCOL2,
        LOOP_INVMIXCOL3,
        LOOP_INVMIXCOL4,
        LOOP_ADDRK,

        INVSUB,
        INVSHIFTROW,
        ADDRK
    } AES_STATE, AES_NEXT_STATE;

    logic [3:0] loop_counter,loop_counter_next;

    assign next_state_out = next_state;

    always_ff @(posedge CLK)
    begin
        if (RESET) begin
            AES_STATE <= WAIT;
            loop_counter <=4'b0;
            state <= 128'b0;

        end

        else begin
            state <= next_state;
            AES_STATE <= AES_NEXT_STATE;
            loop_counter = loop_counter_next;
        end
    end


    //Transition Relations
    always_comb
    begin

        AES_NEXT_STATE = AES_STATE;
        loop_counter_next = loop_counter;

        unique case(AES_STATE)
        WAIT:
        begin
            if(AES_START == 1'b1)
            begin
                AES_NEXT_STATE = KEY_EXPANSION;
                loop_counter_next = 4'b0;
            end
```

```
    else
        AES_NEXT_STATE = WAIT;
end

KEY_EXPANSION:
begin
    loop_counter_next = 4'b0;
    AES_NEXT_STATE = INITIAL_ROUND;
end

INITIAL_ROUND:
    AES_NEXT_STATE = LOOP_INVSHIFTROW;
    //AES_NEXT_STATE = DONE;

LOOP_INVSHIFTROW:
    AES_NEXT_STATE = LOOP_INVSUB;

LOOP_INVSUB:
    AES_NEXT_STATE = LOOP_ADDRK;

LOOP_ADDRK:
    AES_NEXT_STATE = LOOP_INVMIXCOL1;

LOOP_INVMIXCOL1:
    AES_NEXT_STATE = LOOP_INVMIXCOL2;

LOOP_INVMIXCOL2:
    AES_NEXT_STATE = LOOP_INVMIXCOL3;

LOOP_INVMIXCOL3:
    AES_NEXT_STATE = LOOP_INVMIXCOL4;

LOOP_INVMIXCOL4:
begin
    if(loop_counter == 4'd8)
        AES_NEXT_STATE = INVSHIFTROW;
    else
        begin
        loop_counter_next = loop_counter + 4'b1;
        AES_NEXT_STATE = LOOP_INVSHIFTROW;
        end
end

INVSHIFTROW:
    AES_NEXT_STATE = INVSUB;
INVSUB:
    AES_NEXT_STATE = ADDRK;
ADDRK:
    AES_NEXT_STATE = DONE;

DONE:
begin
    if(AES_START == 0)
        AES_NEXT_STATE = WAIT;
    else
        AES_NEXT_STATE = DONE;
end

default:
    AES_NEXT_STATE = WAIT;


endcase
end

//State Contents
always_comb
begin

    next_state = state;
    AES_DONE = 1'b0;

    AES_MSG_DEC = 128'b0;
//    AES_MSG_DEC = state;


        AES_DONE = 0;
    end

LOOP_INVMIXCOL1:
    begin
        mixcolumns_in = state[31:0];
        next_state[31:0] = mixcolumns_out;
        AES_DONE = 0;
    end

LOOP_INVMIXCOL2:
    begin
        mixcolumns_in = state[63:32];
        next_state[63:32] = mixcolumns_out;
        AES_DONE = 0;
    end

LOOP_INVMIXCOL3:
    begin
        mixcolumns_in = state[95:64];
        next_state[95:64] = mixcolumns_out;
        AES_DONE = 0;
    end

LOOP_INVMIXCOL4:
    begin
        mixcolumns_in = state[127:96];
        next_state[127:96] = mixcolumns_out;
        AES_DONE = 0;
    end

LOOP_ADDRK:
    begin
        case (loop_counter)
            4'd0:key = KeySchedule[255:128];
            4'd1:key = KeySchedule[383:256];
            4'd2:key = KeySchedule[511:384];
            4'd3:key = KeySchedule[639:512];
```

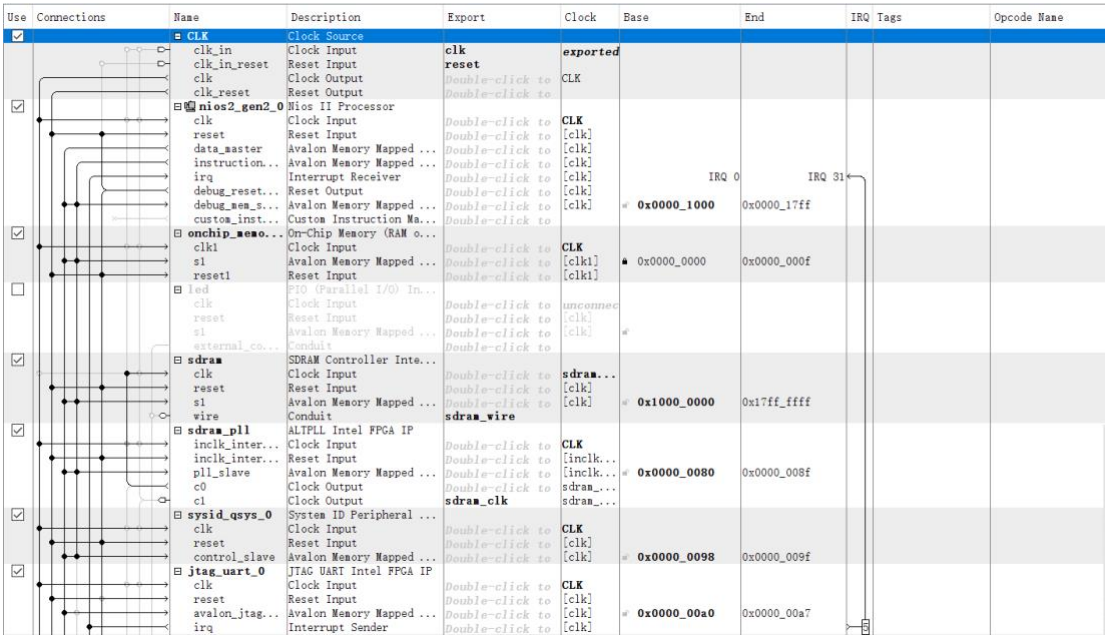
```
mixcolumns_in = 32'b0;
key = 128'b0;

unique case (AES_STATE)

    WAIT:
        begin
            AES_DONE = 0;
        end

    DONE:
        begin
            AES_MSG_DEC = addroundkey_out;
            AES_DONE = 1'b1;
        end

    KEY_EXPANSION:
        begin
            next_state = AES_MSG_ENC;
            AES_DONE = 1'b0;
        end

    INITIAL_ROUND:
        begin
            key = KeySchedule[127:0];
            next_state = addroundkey_out;
            AES_DONE = 1'b0;
        end

    LOOP_INVSUB:
        begin
            next_state = Sub_Out;
            AES_DONE = 0;
        end

    LOOP_INVSHIFTROW:
        begin
            next_state = invshiftrows_out;
```

```
//KeyExpansion
KeyExpansion keyexpansion(.clk(CLK), .Cipherkey(AES_KEY), .KeySchedule(KeySchedule));

// AddRoundKey
AddRoundKey addroundkey(.state(state), .roundKey(key), .out(addroundkey_out));

//InvShiftRows
InvShiftRows invshiftrows(.data_in(state), .data_out(invshiftrows_out));

// InvMixColumns
InvMixColumns invmixcolumns(.in(mixcolumns_in),.out(mixcolumns_out));

// InvSubBytes
InvSubBytes sub0(.clk(CLK), .in(state[7:0]), .out(Sub_Out[7:0]));
InvSubBytes sub1(.clk(CLK), .in(state[15:8]), .out(Sub_Out[15:8]));
InvSubBytes sub2(.clk(CLK), .in(state[23:16]), .out(Sub_Out[23:16]));
InvSubBytes sub3(.clk(CLK), .in(state[31:24]), .out(Sub_Out[31:24]));
InvSubBytes sub4(.clk(CLK), .in(state[39:32]), .out(Sub_Out[39:32]));
InvSubBytes sub5(.clk(CLK), .in(state[47:40]), .out(Sub_Out[47:40]));
InvSubBytes sub6(.clk(CLK), .in(state[55:48]), .out(Sub_Out[55:48]));
InvSubBytes sub7(.clk(CLK), .in(state[63:56]), .out(Sub_Out[63:56]));
InvSubBytes sub8(.clk(CLK), .in(state[71:64]), .out(Sub_Out[71:64]));
InvSubBytes sub9(.clk(CLK), .in(state[79:72]), .out(Sub_Out[79:72]));
InvSubBytes sub10(.clk(CLK), .in(state[87:80]), .out(Sub_Out[87:80]));
InvSubBytes sub11(.clk(CLK), .in(state[95:88]), .out(Sub_Out[95:88]));
InvSubBytes sub12(.clk(CLK), .in(state[103:96]), .out(Sub_Out[103:96]));
InvSubBytes sub13(.clk(CLK), .in(state[111:104]), .out(Sub_Out[111:104]));
InvSubBytes sub14(.clk(CLK), .in(state[119:112]), .out(Sub_Out[119:112]));
InvSubBytes sub15(.clk(CLK), .in(state[127:120]), .out(Sub_Out[127:120]));

endmodule
```

Module: AES.sv
Inputs: CLK, RESET, AES_START, [127:0] AES_KEY, AES_MSG_ENC
Outputs: AES_DONE, [127:0] AES_MSG_DEC, [127:0] next_state_out
Description: It takes the information to execute the decryption process, taking the key and encrypted message and run the decryption algorithm. After execution, it outputs the done signal and decrypted message.
Purpose: It's the main decryption process module.

```
module AddRoundKey (
    input  logic [127:0]  state,
    input  logic [127:0]  roundKey,
    output logic [127:0]  out
);
    always_comb
    begin
        out = state ^ roundKey;
    end

endmodule
```

Module: AddRoundKey.sv
Input: [127:0] state, roundKey
Output: [127:0] out
Description: It XORs the state bytes and the RoundKey together
Purpose: This module serves as the AddRoundKey function in the decryption process.
QSYS Modules:

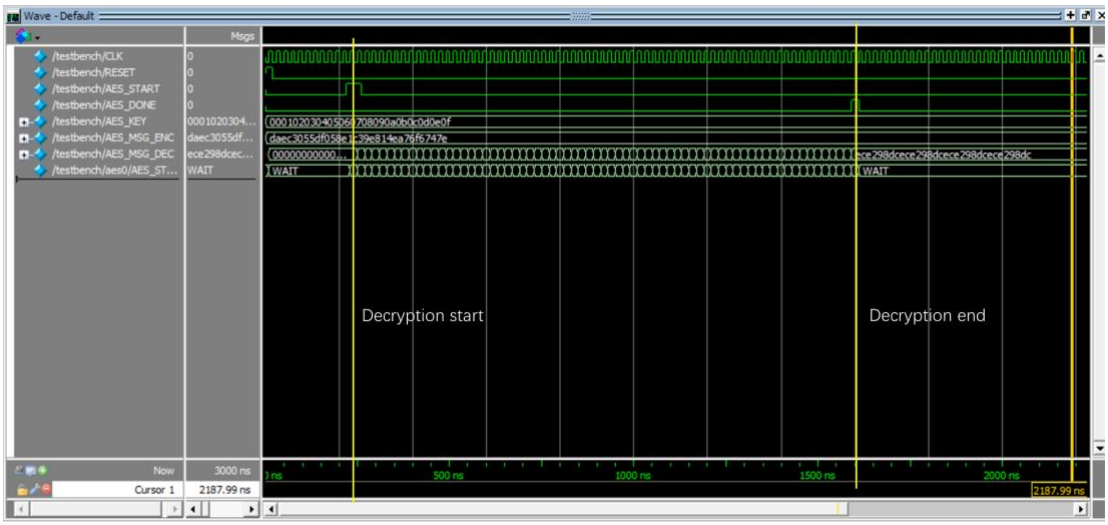


Module: CLK
This is the clock source module that will generate clock signals used in all of other modules
Module: nios2_gen2_0
This is the NIOS-II processor serving as the central controller with the interface with other IO and modules and process the C codes we write.
Module: onchip_memory2_0
This is the on-chip memory module that can be used for storing data. In this lab, we mostly use it to hold the address of our modules.
Module: sdram
This module will allow us to access SDRAM on FPGA.
Module: sdram_pll
This module will generate the phase shift of the clock, so that it can provide a precise clock signal for the SDRAM to read and write data.
Module: sysid_qsys_0
This module is the system ID checker used for ensuring the compatibility between the hardware and software.
Module: jtag_qsys_0
This allows for terminal access for use in software debugging.
Module: AES
This is the AES core to execute the decryption process.
Module: TIMER
This acts as a clock synchronizer for the software w.r.t hardware components.

III. Annotated Simulation of the AES decryptor

1) Code of testbench.sv

```
module testbench();

timeunit 10ns;

timeprecision 1ns;
logic CLK;
logic RESET;
logic AES_START;
logic AES_DONE;
logic [127:0] AES_KEY;
logic [127:0] AES_MSG_ENC;
logic [127:0] AES_MSG_DEC;
logic [127:0] next_state_out;
AES aes0(.*);

always begin : CLOCK_GENERATION
#1 CLK = ~CLK;
end

initial begin : CLOCK_INITIALIZATION
   CLK = 0;
   end

initial begin : TEST_VECTORS
    AES_START = 0;
    RESET = 1;
    AES_KEY = 128'h000102030405060708090a0b0c0d0e0f;
    AES_MSG_ENC = 128'hdaec3055df058e1c39e814ea76f6747e;

    #2 RESET = 0;
    #15 AES_START = 1;
    #4   AES_START = 0;
 end

endmodule
```



2) Result of testbench

The figure shown above is the testbench result of AES module. AES_KEY signal is the key for decryption, which is 000102030405060708090a0b0c0d0e0f, AES_MSG_ENC is the input of the AES module, which is the encrypted result from the software encryption, which is daec3055df058e1c39e814ea76f6747e. And AES_MSG_DEC is the decode result from AES module, after decryption, the result is ece298dcece298dcece298dcece298dc, which is the same as the input message to the encryption function. AES_STATE is the inner decryption state for AES module. We annotate the start and end point in the figure.

IV. Post-Lab Questions

1) Design Resources and Statistics Table

LUT	5913
DSP	0
Memory(BRAM)	126080
Flip-Flop	2873
Frequency	143.2 MHz
Static Power	103.91 mW
Dynamic Power	1.02 mW
Total Power	183.17 mW

2) Answers to the post-lab questions

(a) We expect hardware-based decryption to be faster than software encryption. Our benchmark results, shown below, which are in line with our expectations.

```
Software Encryption Speed: 0.499875 KB/s
Hardware Encryption Speed: 200.000000 KB/s
```

(b) Because we are only allowed to use one InvMixColumns module in this lab. In this case, we need four cycles to complete an InvMixColumns operation because it only processes one column at a time. Therefore, we can extend this module to handle the entire matrix. Specifically, the submodule here can process a column so that the entire module can process the entire matrix in parallel. In this case, it only takes one cycle to complete the InvMixColumn operation and saves a lot of time. In addition, the InvSubBytes module and the AddRoundKey module can be combined into one module. Since InvSubBytes is a substitution operation for each element in the matrix, and AddRoundKey will xor a specific value to the position of the matrix element based on its position, we can combine the two into one : out[i] <= sub_value[i] ^ roundkey[i]. In this case, the InvSubBytes and AddRoundKey operations require only one state. It will save a lot of time.

V. Conclusion

Our design succeeded in creating a state machine that handled decryption, wrote it to the register file and displayed the first and last two bytes on the hex displays. We were able to display the correct encrypted and decrypted message in the NIOS terminal and got expected benchmark values. We were not, however, able to run consecutive decryptions correctly. We suspect this to be an error in changing one of the values needed to run the steps correctly. We would need to run the code in simulation and confirm that the initial values are correct, then check keyExpansion to make sure that we are not altering the con table when expanding keys.

A note explaining how to convert the char arrays to the plaintext and vice versa would have been helpful, as well as converting the con array to a format we could use would have been helpful. We figured out that we needed to right shift 24 bits in keyExpansion but it took us a fair amount of time to figure out. Other than that, nothing was horribly unclear. The intermediate steps in the given .txt file was incredibly helpful and helped us get our demo points. One for the other test case could be helpful.