```
/************************************************************
--  HyRAL128_ENC
--  Programed by K.I.Technology
--  2009.12.30
--  File Name: HyRAL128_main.v
************************************************************/

/*
----------------------------- 構成 -----------------------------
HyRAL128_main--+-- intface
               +-- keygen
               +-- bkeygen
               +-- skeygen
               +-- encrypt
               +-- gfunc
               +-- lffunc
               +-- decrypt
               +-- gifunc
               +-- lfifunc
               +-- ffunc    -- sbox_rom
-------------------------------------------------------------

----------------------------- 機能 -----------------------------
本モジュールは、128bit入力データを暗号化および復号化します。
-------------------------------------------------------------

*/

/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   HyRAL128_main( CLK,     RST,
                        DATA,    ASTRB,   DSTRB,
                        KEYEND,  EEND,    DEND,  RDATA
                      );

// Input/Output
input         CLK;                    // System Clock
input         RST;                   // System Reset

input  [ 31:0] DATA;                    // WriteData Bus
input         ASTRB, DSTRB;              // Strobe

output        KEYEND;                  // SUBKEY GENERAE END
output        EEND;                   // Encord END
output        DEND;                    // Decord END
output [ 31:0] RDATA;                    // Read DATA

wire          SDSW;                   // Single Key(128bit), Double Key(192,256bit) Mode
wire          BSTART;                  // KeyGen Start
wire          CSTART;                  // Encrypt Start
wire          CGSTART;                  // G function start
wire          DGSTART;                  // G INVERSE function start
wire          KSTART;                  // BKEYGEN START
```

```
wire        BKGSTART;                 // G function start
wire        LFSTART;                  // F function start
wire        LFISTART;                 // F Inverse Start
wire        FSTART;                   // f function Start
wire        GFSTART;                  // G function Start
wire        DSTART;                   // Decrypt Start
wire        DLFISTART;                // F inverse Start

wire  [127:0] OK1, OK2;               // Original Key1(128mode or 256mode), Original
Key2(256mode only)
wire  [127:0] PT;                     // Plaintext

wire  [127:0] KM1, KM2, KM3, KM4;         // Key Material
wire  [127:0] SKM1, SKM2, SKM3, SKM4;     // Key Material
wire  [127:0] IK1, IK2, IK3, IK4, IK5, IK6; // I Key
wire  [127:0] RK1, RK2, RK3, RK4,
          RK5, RK6, RK7, RK8, RK9;    // R Key
wire  [127:0] Y2, Z2, W2;             // Key Material
wire  [ 31:0] CST;
wire        BEND;
wire  [127:0] KCONST,OK;
wire        BKGREQ, CGREQ;
wire        BKGSEL, CGSEL, DGSEL, LFSEL, LFISEL;
wire  [127:0] BKGDATA, CGDATA, DGDATA;

wire        GEND, GIEND;
wire  [127:0] GODATA, GIODATA;

wire  [127:0] LFDATA, IK, LFODATA, LFIIK, LFIODATA, DLFDATA;
wire        LFEND, LFIEND;

wire  [ 31:0] GFDATA, GIFDATA, LFIDATA;
wire  [  2:0] GAT, LFAT, GIAT, LFIAT;

wire        FEND;
wire  [ 31:0] FODATA;
wire  [ 31:0] FDATA;
wire        GFREQ, LFREQ, GIFREQ, LFIREQ;
wire  [127:0] ENC_DATA, DEC_DATA;

// -------------------- interface
intface   intface  ( .CLK( CLK ),
                .RST( RST ),
                .ASTRB( ASTRB ),
                .DATA( DATA ),
                .DSTRB( DSTRB),
                .PT( PT ),
                .OK1( OK1 ),
                .OK2( OK2 ),
                .SDSW( SDSW ),
                .BSTART( BSTART ),
                .CSTART( CSTART ),
                .DSTART( DSTART ),
                .CST( CST ),
```

```
                .Y2( Y2 ),
                .Z2( Z2 ),
                .W2( W2 ),
                .ENC( ENC_DATA ),
                .EEND( EEND ),
                .DEC( DEC_DATA ),
                .DEND( DEND ),
                .RDATA( RDATA )
              );

// -------------------- KeyGen Block
keygen    keygen   ( .CLK ( CLK ),
                .RST ( RST ),
                .SDSW( SDSW ),
                .BSTART(BSTART),
                .OK1 ( OK1 ),
                .OK2 ( OK2 ),
                .CONST1( Y2 ),  // 2009_12_16
                .CONST2( Z2 ),  // 2009_12_16
                .CONST3( W2 ),  // 2009_12_16
                .IKM1 ( SKM1 ),
                .IKM2 ( SKM2 ),
                .IKM3 ( SKM3 ),
                .IKM4 ( SKM4 ),
                .BEND ( BEND ),
                .KM1 ( KM1 ),
                .KM2 ( KM2 ),
                .KM3 ( KM3 ),
                .KM4 ( KM4 ),
                .KSTART( KSTART ),
                .CONST( KCONST ),
                .OK( OK ),
                .KEYEND( KEYEND )
              );

bkeygen  bkeygen   ( .CLK  ( CLK ),
                .RST  ( RST ),
                .KSTART(KSTART),
                .CONST( KCONST ),
                .OK   ( OK ),
                .KM1  ( SKM1 ),
                .KM2  ( SKM2 ),
                .KM3  ( SKM3 ),
                .KM4  ( SKM4 ),
                .BEND ( BEND ),
                .GREQ ( BKGREQ ),
                .GSEL ( BKGSEL ),
                .GDATA( BKGDATA ),
                .GSTART( BKGSTART ),
                .GEND ( GEND ),
                .GODATA( GODATA )
            );

skeygen skeygen    ( .CLK ( CLK ),
```

```
                .RST ( RST ),
                .SDSW( SDSW ),
                .KM1 ( KM1 ),
                .KM2 ( KM2 ),
                .KM3 ( KM3 ),
                .KM4 ( KM4 ),
                .IK1 ( IK1 ),
                .IK2 ( IK2 ),
                .IK3 ( IK3 ),
                .IK4 ( IK4 ),
                .IK5 ( IK5 ),
                .IK6 ( IK6 ),
                .RK1 ( RK1 ),
                .RK2 ( RK2 ),
                .RK3 ( RK3 ),
                .RK4 ( RK4 ),
                .RK5 ( RK5 ),
                .RK6 ( RK6 ),
                .RK7 ( RK7 ),
                .RK8 ( RK8 ),
                .RK9 ( RK9 )
              );

// -------------------- Encrypt
encrypt encrypt    ( .CLK ( CLK ),
                .RST ( RST ),
                .SDSW( SDSW ),
                .CSTART( CSTART ),
                .IK1 ( IK1 ),
                .IK2 ( IK2 ),
                .IK3 ( IK3 ),
                .IK4 ( IK4 ),
                .IK5 ( IK5 ),
                .IK6 ( IK6 ),
                .RK1 ( RK1 ),
                .RK2 ( RK2 ),
                .RK3 ( RK3 ),
                .RK4 ( RK4 ),
                .RK5 ( RK5 ),
                .RK6 ( RK6 ),
                .RK7 ( RK7 ),
                .RK8 ( RK8 ),
                .RK9 ( RK9 ),
                .PT  ( PT ),
                .CT  ( ENC_DATA ),
                .CEND( EEND ),
                .GREQ( CGREQ ),
                .GSTART( CGSTART ),
                .GSEL( CGSEL ),
                .GDATA( CGDATA ),
                .GEND( GEND ),
                .GODATA( GODATA ),
                .LFSTART( LFSTART ),
                .LFSEL( LFSEL ),
```

```
                .LFDATA( LFDATA ),
                .IK( IK ),
                .LFEND( LFEND ),
                .LFODATA( LFODATA )
               );

// -------------------- Decrypt
decrypt decrypt    ( .CLK ( CLK ),
                .RST ( RST ),
                .SDSW( SDSW ),
                .CSTART( DSTART ),
                .IK1 ( IK1 ),
                .IK2 ( IK2 ),
                .IK3 ( IK3 ),
                .IK4 ( IK4 ),
                .IK5 ( IK5 ),
                .IK6 ( IK6 ),
                .RK1 ( RK1 ),
                .RK2 ( RK2 ),
                .RK3 ( RK3 ),
                .RK4 ( RK4 ),
                .RK5 ( RK5 ),
                .RK6 ( RK6 ),
                .RK7 ( RK7 ),
                .RK8 ( RK8 ),
                .RK9 ( RK9 ),
                .PT  ( PT ),
//                .PT  ( ENC_DATA ),
                .CT  ( DEC_DATA ),
                .CEND( DEND ),
                .GSTART( DGSTART ),
                .GSEL( DGSEL ),
                .GDATA( DGDATA ),
                .GEND( GIEND ),
                .GODATA( GIODATA ),
                .LFSTART( DLFISTART ),
                .LFSEL( LFISEL ),
                .LFDATA( DLFDATA ),
                .IK( LFIIK ),
                .LFEND( LFIEND ),
                .LFODATA( LFIODATA )
               );

// -------------------- G inverse function
gifunc  gifunc ( .CLK( CLK ),
              .RST( RST ),
              .GSEL  ( DGSEL ),
              .GSTART( DGSTART ),
              .XDATA ( DGDATA ),
              .GEND( GIEND ),
              .YDATA( GIODATA ),
              .FSTART( GIFSTART ),
              .FDATA( GIFDATA ),
              .AT( GIAT ),
```

```
              .FREQ( GIFREQ ),
              .FEND( FEND ),
              .FODATA( FODATA )
            );

// -------------------- F inverse function
lfifunc lfifunc( .CLK( CLK ),
              .RST( RST ),
              .LFSTART( DLFISTART ),
              .FSEL( LFISEL ),
              .XDATA( DLFDATA ),
              .IK( LFIIK ),
              .LFEND( LFIEND ),
              .YDATA( LFIODATA ),
              .FSTART( LFISTART ),
              .FDATA( LFIDATA ),
              .AT( LFIAT ),
              .FREQ( LFIREQ ),
              .FEND( FEND ),
              .FODATA( FODATA )
            );

// -------------------- G function
gfunc  gfunc ( .CLK( CLK ),
            .RST( RST ),
            .GSEL  ( MUX2to1DATA1(  {BKGREQ, CGREQ}, BKGSEL,   CGSEL) ),
            .GSTART( MUX2to1DATA1(  {BKGREQ, CGREQ}, BKGSTART, CGSTART) ),
            .XDATA ( MUX2to1DATA128({BKGREQ, CGREQ}, BKGDATA,  CGDATA) ),
            .GEND( GEND ),
            .YDATA( GODATA ),
            .FSTART( GFSTART ),
            .FDATA( GFDATA ),
            .AT( GAT ),
            .FREQ( GFREQ ),
            .FEND( FEND ),
            .FODATA( FODATA )
          );

// -------------------- F function
lffunc lffunc( .CLK( CLK ),
            .RST( RST ),
            .LFSTART( LFSTART ),
            .FSEL( LFSEL ),
            .XDATA( LFDATA ),
            .IK( IK ),
            .LFEND( LFEND ),
            .YDATA( LFODATA ),
            .FSTART( FSTART ),
            .FDATA( FDATA ),
            .AT( LFAT ),
            .FREQ( LFREQ ),
            .FEND( FEND ),
            .FODATA( FODATA )
          );
```

```
// -------------------- f function
ffunc ffuncg( .CLK(CLK),
            .RST(RST),
            .FSTART(MUX4to1DATA1(  {GFREQ, LFREQ, GIFREQ, LFIREQ}, GFSTART, FSTART,
GIFSTART, LFISTART)),
            .XDATA( MUX4to1DATA32( {GFREQ, LFREQ, GIFREQ, LFIREQ}, GFDATA,  FDATA,
 GIFDATA,  LFIDATA) ),
            .KEY(32'h0),
            .AT(MUX4to1DATA3( {GFREQ, LFREQ, GIFREQ, LFIREQ}, GAT, LFAT, GIAT, LFIAT))
,
            .CST1(CST[ 7: 0]),
            .CST2(CST[15: 8]),
            .CST3(CST[23:16]),
            .CST4(CST[31:24]),
            .FEND(FEND),
            .YDATA(FODATA)
          );

// Multiplex 4 to 1

function [31:0] MUX4to1DATA32;
input [ 3:0] sela; // expected sela = {areq, breq, creq, dreq};
input [31:0] adata, bdata, cdata, ddata;

  case ( sela )
       4'h8 : MUX4to1DATA32 = adata;
       4'h4 : MUX4to1DATA32 = bdata;
       4'h2 : MUX4to1DATA32 = cdata;
       4'h1 : MUX4to1DATA32 = ddata;
    default : MUX4to1DATA32 = adata;
  endcase

endfunction

// Multiplex 4 to 1
function MUX4to1DATA1;
input [ 3:0] selb; // expected selb = { ereq, freq, greq, hreq};
input edata, fdata, gdata, hdata;

  case ( selb )
       4'h8 : MUX4to1DATA1 = edata;
       4'h4 : MUX4to1DATA1 = fdata;
       4'h2 : MUX4to1DATA1 = gdata;
       4'h1 : MUX4to1DATA1 = hdata;
    default : MUX4to1DATA1 = edata;
  endcase

endfunction

// Multiplex 2 to 1
function [127:0] MUX2to1DATA128;
input [  1:0] selc; // expected selc = {ireq, jreq};
input [127:0] idata, jdata;
```

```
  case ( selc )
       2'h2 : MUX2to1DATA128 = idata;
       2'h1 : MUX2to1DATA128 = jdata;
     default : MUX2to1DATA128 = idata;
  endcase

endfunction

// Multiplex 4 to 1
function [2:0] MUX4to1DATA3;
input [ 3:0] seld;  // expected seld = { kreq, lreq, mreq, nreq};
input [ 2:0] kdata, ldata, mdata, ndata;

  case ( seld )
       4'h8 : MUX4to1DATA3 = kdata;
       4'h4 : MUX4to1DATA3 = ldata;
       4'h2 : MUX4to1DATA3 = mdata;
       4'h1 : MUX4to1DATA3 = ndata;
     default : MUX4to1DATA3 = kdata;
  endcase

endfunction

// Multiplex 2 to 1
function MUX2to1DATA1;
input [ 1:0] sele; // expected selb = { ereq, freq, greq, hreq};
input odata, pdata;

  case ( sele )
       2'h2 : MUX2to1DATA1 = odata;
       2'h1 : MUX2to1DATA1 = pdata;
     default : MUX2to1DATA1 = odata;
  endcase

endfunction

endmodule


/*************************************************************
--  HyRAL128bit
--  Programed by K.I.Technology
--  2009.12.30
--  File Name: intface.v
*************************************************************/

*/
------------------------------ 機能 ------------------------------
本モジュールは、インターフェース機能を有しライトバス、リードバス
を兼ね備えた専用インターフェースです。
------------------------------------------------------------------

*/
```

```
/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   intface( CLK,      RST,
                  ASTRB,    DATA,     DSTRB,
                  PT,       OK1,      OK2,
                  SDSW,     BSTART,   CSTART,   DSTART,
                  CST,      Y2,       Z2,       W2,
                  ENC,      EEND,     DEC,      DEND,
                  RDATA
                );

// Input/Output
input         CLK;                        // System Clock
input         RST;                        // System Reset
input  [ 31:0] DATA;
input         ASTRB, DSTRB;                    // F1 or F2 SEL 0=F1, 1=F2
input         EEND, DEND;
input  [127:0] ENC, DEC;

output [127:0] PT;
output [127:0] OK1, OK2;

output         SDSW, BSTART, CSTART, DSTART;
output [ 31:0] CST;
output [127:0] Y2,Z2,W2;
output [ 31:0] RDATA;

reg    [  7:0] ADRS;
reg    [127:0] PT, OK1, OK2;
reg            SDSW, BSTART, CSTART, DSTART;
reg    [127:0] Y2, Z2, W2;
reg    [127:0] ENCDATA, DECDATA;
reg    [ 31:0] RDATA;

//reg    [ 31:0] CST;

always @(posedge CLK or posedge RST)
  if(RST)
    ADRS  <= 4'h0;
  else if( ASTRB )
    ADRS  <= DATA;

always @(posedge CLK or posedge RST) begin
  if(RST) begin
    PT   <= 128'h0;
    OK1  <= 128'h0;
    OK2  <= 128'h0;
    SDSW <= 1'h0;
//     CST  <= 32'h8844_2211;    // for extened
    Y2   <= 128'h628C_CDA0_3B15_65C1_3BAD_2D4F_B880_6AC5;
    Z2   <= 128'hF925_1A23_65CD_3C2E_8066_CBBB_FE31_6B7B;
    W2   <= 128'h5DE2_8625_656B_71FF_9FFB_1E12_EEF1_27F5;
```

```
  end
  else if( DSTRB )
    case( ADRS )
      8'h04 :  SDSW <= DATA[0];
      8'h08 :  PT [ 31: 0]  <= DATA;
      8'h0C :  PT [ 63:32]  <= DATA;
      8'h10 :  PT [ 95:64]  <= DATA;
      8'h14 :  PT [127:96]  <= DATA;
      8'h18 :  OK1[ 31: 0]  <= DATA;
      8'h1C :  OK1[ 63:32]  <= DATA;
      8'h20 :  OK1[ 95:64]  <= DATA;
      8'h24 :  OK1[127:96]  <= DATA;
      8'h28 :  OK2[ 31: 0]  <= DATA;
      8'h2C :  OK2[ 63:32]  <= DATA;
      8'h30 :  OK2[ 95:64]  <= DATA;
      8'h34 :  OK2[127:96]  <= DATA;
//      8'h38 :  CST         <= DATA;    // for extened
//      4'h3C :  K           <= DATA;  // spare
      8'h40 :  Y2 [ 31:0]   <= DATA;
      8'h44 :  Y2 [ 63:32]  <= DATA;
      8'h48 :  Y2 [ 95:64]  <= DATA;
      8'h4C :  Y2 [127:96]  <= DATA;
      8'h50 :  Z2 [ 31:0]   <= DATA;
      8'h54 :  Z2 [ 63:32]  <= DATA;
      8'h58 :  Z2 [ 95:64]  <= DATA;
      8'h5C :  Z2 [127:96]  <= DATA;
      8'h60 :  W2 [ 31:0]   <= DATA;
      8'h64 :  W2 [ 63:32]  <= DATA;
      8'h68 :  W2 [ 95:64]  <= DATA;
      8'h6C :  W2 [127:96]  <= DATA;
    endcase
end

always @(posedge CLK or posedge RST) begin
  if(RST) begin
    BSTART <= 1'h0;
    CSTART <= 1'h0;
    DSTART <= 1'h0;
  end
  else begin
    BSTART <= DSTRB & (ADRS==4'h0) & DATA[0];
    CSTART <= DSTRB & (ADRS==4'h0) & DATA[1];
    DSTART <= DSTRB & (ADRS==4'h0) & DATA[2];
  end
end

// Fixed Value
assign CST  = 32'h8844_2211;

always @(posedge CLK or posedge RST) begin
  if(RST)
    ENCDATA <= 128'h0;
  else if( EEND )
    ENCDATA <= ENC;
```

```
end

always @(posedge CLK or posedge RST) begin
  if(RST)
    DECDATA <= 128'h0;
  else if( DEND )
    DECDATA <= DEC;
end

// Data Load
always @(posedge CLK or posedge RST) begin
  if(RST)
    RDATA <= 32'h0;
  else
    case( ADRS )
      8'h04 :  RDATA <= { 31'h0, SDSW};
      8'h08 :  RDATA <= PT [ 31: 0];
      8'h0C :  RDATA <= PT [ 63:32];
      8'h10 :  RDATA <= PT [ 95:64];
      8'h14 :  RDATA <= PT [127:96];
      8'h18 :  RDATA <= OK1[ 31: 0];
      8'h1C :  RDATA <= OK1[ 63:32];
      8'h20 :  RDATA <= OK1[ 95:64];
      8'h24 :  RDATA <= OK1[127:96];
      8'h28 :  RDATA <= OK2[ 31: 0];
      8'h2C :  RDATA <= OK2[ 63:32];
      8'h30 :  RDATA <= OK2[ 95:64];
      8'h34 :  RDATA <= OK2[127:96];
//      8'h38 :  RDATA <= CST;    // for extened
//      4'h3C :  RDATA <= K;  // spare
      8'h40 :  RDATA <= Y2 [ 31: 0];
      8'h44 :  RDATA <= Y2 [ 63:32];
      8'h48 :  RDATA <= Y2 [ 95:64];
      8'h4C :  RDATA <= Y2 [127:96];
      8'h50 :  RDATA <= Z2 [ 31: 0];
      8'h54 :  RDATA <= Z2 [ 63:32];
      8'h58 :  RDATA <= Z2 [ 95:64];
      8'h5C :  RDATA <= Z2 [127:96];
      8'h60 :  RDATA <= W2 [ 31: 0];
      8'h64 :  RDATA <= W2 [ 63:32];
      8'h68 :  RDATA <= W2 [ 95:64];
      8'h6C :  RDATA <= W2 [127:96];
      8'h70 :  RDATA <= ENCDATA[ 31: 0];
      8'h74 :  RDATA <= ENCDATA[ 63:32];
      8'h78 :  RDATA <= ENCDATA[ 95:64];
      8'h7C :  RDATA <= ENCDATA[127:96];
      8'h80 :  RDATA <= DECDATA[ 31: 0];
      8'h84 :  RDATA <= DECDATA[ 63:32];
      8'h88 :  RDATA <= DECDATA[ 95:64];
      8'h8C :  RDATA <= DECDATA[127:96];
    endcase
end
```

```
endmodule


/************************************************************
--  HyRAL128bit
--  Programed by K.I.Technology
--  2009.12.01
--  File Name: keygen.v
************************************************************/

/*
----------------------------- 機能 -----------------------------
本モジュールは、128bit keygeneration controlを行います。
----------------------------------------------------------------

*/

/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   keygen ( CLK,    RST,    SDSW,   BSTART,
                  OK1,    OK2,
                  CONST1, CONST2, CONST3,
                  IKM1,   IKM2,   IKM3,   IKM4,
                  BEND,
                  KM1,    KM2,    KM3,    KM4,
                  KSTART, CONST,  OK,     KEYEND
                );

// Input/Output
input         CLK;                      // System Clock
input         RST;                      // System Reset
input         SDSW;
input         BSTART;

input         BEND;
input  [127:0] IKM1, IKM2, IKM3, IKM4;

input  [127:0] OK1, OK2;
input  [127:0] CONST1, CONST2, CONST3;

output [127:0] KM1, KM2, KM3, KM4;
output         KEYEND;
output         KSTART;
output [127:0] CONST;
output [127:0] OK;

reg    [127:0] SKM1, SKM2, SKM3, SKM4;
reg    [127:0] KM1,  KM2,  KM3,  KM4;
reg    [  3:0] kstate;
reg            KSTART,KEYEND;
reg    [127:0] CONST;
reg    [127:0] OK;
```

```
parameter KIDLE = 4'h0,
          KSTATE1 = 4'h1,
          KSTATE2 = 4'h2;

always @(posedge CLK or posedge RST) begin
  if(RST) begin
    kstate <= KIDLE;
    KSTART <= 1'h0;
    CONST  <= 128'h0;
    OK     <= 128'h0;
    KEYEND <= 1'h0;
    SKM1   <= 128'h0;
    SKM2   <= 128'h0;
    SKM3   <= 128'h0;
    SKM4   <= 128'h0;
    KM1    <= 128'h0;
    KM2    <= 128'h0;
    KM3    <= 128'h0;
    KM4    <= 128'h0;
  end
  else
    case ( kstate )
    KIDLE   : if( BSTART ) begin
                kstate  <= KSTATE1;
                KSTART  <= 1'h1;
                CONST   <= (SDSW==1'h0) ? CONST1 : CONST2;
                OK      <= OK1;
                KEYEND <= 1'h0;
              end
              else
                KEYEND <= 1'h0;

    KSTATE1 : if( BEND )
                if( SDSW==1'h1) begin
                  kstate <= KSTATE2;
                  KSTART <= 1'h1;
                  CONST  <= CONST3;
                  OK     <= OK2;
                  SKM1   <= IKM1;
                  SKM2   <= IKM2;
                  SKM3   <= IKM3;
                  SKM4   <= IKM4;
                end
                else  begin
//                  else if( SDSW==1'h0 ) begin
                  kstate <= KIDLE;
                  KM1    <= IKM1;
                  KM2    <= IKM2;
                  KM3    <= IKM3;
                  KM4    <= IKM4;
                  KEYEND <= 1'h1;
                end
              else
                KSTART <= 1'h0;
```

```
    KSTATE2 : if( BEND ) begin
                kstate <= KIDLE;
                KM1    <= SKM1 ^ IKM1;
                KM2    <= SKM2 ^ IKM2;
                KM3    <= SKM3 ^ IKM3;
                KM4    <= SKM4 ^ IKM4;
                KEYEND <= 1'h1;
              end
              else
                KSTART <= 1'h0;
    endcase
end



endmodule


/************************************************************
--  HyRAL128_ENC
--  Programed by K.I.Technology
--  2009.12.01
--  File Name: bkeygen.v
************************************************************/

/*
---------------------------- 構成 ----------------------------
HyRAL128_ENCmain
--------------------------------------------------------------

---------------------------- 機能 ----------------------------
本モジュールは、128bit HyRAL暗号化機能を有します。
--------------------------------------------------------------

*/

/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   bkeygen ( CLK,    RST,
                   OK,     CONST,  KSTART,
                   KM1,    KM2,    KM3,    KM4,
                   BEND,
                   GREQ,   GSEL,   GDATA,
                   GEND,   GODATA, GSTART
                 );

// Input/Output
input         CLK;                    // System Clock
input         RST;                    // System Reset
input         KSTART;

input  [127:0] OK;
```

```
input  [127:0] CONST;

input          GEND;
input  [127:0] GODATA;

output [127:0] KM1, KM2, KM3, KM4;
output         BEND;

output         GREQ;
output         GSEL;
output [127:0] GDATA;
output         GSTART;

reg    [127:0] KM1, KM2, KM3, KM4;
reg    [127:0] GDATA;
reg    [  7:0] bstate;
reg            GSEL, GSTART;
reg            GREQ;
reg            BEND;

wire   [127:0] GODATA;

parameter      BIDLE   = 8'h0,
               BSTATE1 = 8'h1,
               BSTATE2 = 8'h2,
               BSTATE3 = 8'h3,
               BSTATE4 = 8'h4,
               BSTATE5 = 8'h5;

always @(posedge CLK or posedge RST) begin
  if(RST) begin
    bstate <= BIDLE;
    GSTART <= 1'h0;
    GDATA  <= 128'h0;
    GSEL   <= 1'h0;
    GREQ   <= 1'h0;
    BEND   <= 1'h0;
  end
  else
    case ( bstate )
    BIDLE   : if( KSTART ) begin
                bstate <= BSTATE1;
                GSTART <= 1'h1;
                GDATA  <= CONST ^ OK;
                GSEL   <= 1'h0;          // G1
                GREQ   <= 1'h1;
                BEND   <= 1'h0;
              end
              else
                BEND   <= 1'h0;

    BSTATE1 : if( GEND ) begin
                bstate <= BSTATE2;
                GSTART <= 1'h1;
```

```
            GDATA  <= OK ^ GODATA;
            GSEL   <= 1'h1;          // G2
          end
          else
            GSTART <= 1'h0;

    BSTATE2 : if( GEND ) begin
            bstate <= BSTATE3;
            GSTART <= 1'h1;
            GDATA  <= OK ^ GODATA;
            GSEL   <= 1'h0;          // G1
            KM1    <= GODATA;
          end
          else
            GSTART <= 1'h0;

    BSTATE3 : if( GEND ) begin
            bstate <= BSTATE4;
            GSTART <= 1'h1;
            GDATA  <= OK ^ GODATA;
            GSEL   <= 1'h1;          // G2
            KM3    <= GODATA;
          end
          else
            GSTART <= 1'h0;

    BSTATE4 : if( GEND ) begin
            bstate <= BSTATE5;
            GSTART <= 1'h1;
            GDATA  <= OK ^ GODATA;
            GSEL   <= 1'h0;          // G1
            KM2    <= GODATA;
          end
          else
            GSTART <= 1'h0;

    BSTATE5 : if( GEND ) begin
            bstate <= BIDLE;
            KM4    <= GODATA;
            GREQ   <= 1'h0;
            BEND   <= 1'h1;
          end
          else
            GSTART <= 1'h0;
    endcase
end

endmodule


/************************************************************
--  HyRAL128bit
--  Programed by K.I.Technology
--  2009.12.01
```

```
--  File Name: skeygen.v
************************************************************/

/*
----------------------------- 機能 -----------------------------
本モジュールは、128bit HyRAL暗号化で用いられるSUBKEY生成機能を
有します。
------------------------------------------------------------

*/

/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   skeygen ( CLK,    RST,    SDSW,
                   KM1,    KM2,    KM3,    KM4,
                   IK1,    IK2,    IK3,    IK4,    IK5,    IK6,
                   RK1,    RK2,    RK3,    RK4,    RK5,    RK6,
                   RK7,    RK8,    RK9
                 );

// Input/Output
input         CLK;                        // System Clock
input         RST;                        // System Reset
input         SDSW;
input  [127:0] KM1, KM2, KM3, KM4;
output [127:0] IK1, IK2, IK3, IK4, IK5, IK6;
output [127:0] RK1, RK2, RK3, RK4, RK5, RK6, RK7, RK8, RK9;

reg    [127:0] IK1, IK2, IK3, IK4, IK5, IK6;
reg    [127:0] RK1, RK2, RK3, RK4, RK5, RK6, RK7, RK8, RK9;


always @(posedge CLK or posedge RST) begin
  if(RST) begin
    IK1 <= 128'h0;
    IK2 <= 128'h0;
    IK3 <= 128'h0;
    IK4 <= 128'h0;
    IK5 <= 128'h0;
    IK6 <= 128'h0;
    RK1 <= 128'h0;
    RK2 <= 128'h0;
    RK3 <= 128'h0;
    RK4 <= 128'h0;
    RK5 <= 128'h0;
    RK6 <= 128'h0;
    RK7 <= 128'h0;
    RK8 <= 128'h0;
    RK9 <= 128'h0;
  end
  else if(SDSW) begin  // for double key mode
    IK1 <= {KM1[95:64],KM1[63:32],KM1[31:0],KM1[127:96]} ^ {KM4[127:96],KM4[31:0]
,KM4[63:32],KM4[95:64]};
```

```
    IK2 <= {KM2[31:0],KM2[63:32],KM2[95:64],KM2[127:96]} ^ {KM3[127:96],KM3[95:6
4],KM3[63:32],KM3[31:0]};
    IK3 <= {KM2[127:96],KM2[31:0],KM2[63:32],KM2[95:64]} ^ {KM3[95:64],KM3[63:32]
,KM3[31:0],KM3[127:96]};
    IK4 <= {KM2[127:96],KM2[95:64],KM2[63:32],KM2[31:0]} ^ {KM3[63:32],KM3[31:0],
KM3[127:96],KM3[95:64]};
    IK5 <= {KM2[31:0],KM2[127:96],KM2[95:64],KM2[63:32]} ^ {KM3[31:0],KM3[127:96]
,KM3[95:64],KM3[63:32]};
    IK6 <= {KM1[63:32],KM1[31:0],KM1[127:96],KM1[95:64]} ^ {KM4[127:96],KM4[95:6
4],KM4[63:32],KM4[31:0]};
    RK1 <= {KM3[31:0],KM3[63:32],KM3[95:64],KM3[127:96]} ^ {KM4[127:96],KM4[95:6
4],KM4[63:32],KM4[31:0]};
    RK2 <= {KM1[127:96],KM1[31:0],KM1[63:32],KM1[95:64]} ^ {KM2[95:64],KM2[63:32
],KM2[31:0],KM2[127:96]};
    RK3 <= {KM3[127:96],KM3[95:64],KM3[63:32],KM3[31:0]} ^ {KM4[63:32],KM4[31:0],
KM4[127:96],KM4[95:64]};
    RK4 <= {KM3[31:0],KM3[127:96],KM3[95:64],KM3[63:32]} ^ {KM4[31:0],KM4[127:96
],KM4[95:64],KM4[63:32]};
    RK5 <= {KM1[127:96],KM1[95:64],KM1[63:32],KM1[31:0]} ^ {KM4[31:0],KM4[63:32],
KM4[95:64],KM4[127:96]};
    RK6 <= {KM1[31:0],KM1[127:96],KM1[95:64],KM1[63:32]} ^ {KM2[31:0],KM2[127:96
],KM2[95:64],KM2[63:32]};
    RK7 <= {KM1[127:96],KM1[95:64],KM1[63:32],KM1[31:0]} ^ {KM2[63:32],KM2[31:0],
KM2[127:96],KM2[95:64]};
    RK8 <= {KM3[127:96],KM3[31:0],KM3[63:32],KM3[95:64]} ^ {KM4[95:64],KM4[63:32
],KM4[31:0],KM4[127:96]};
    RK9 <= {KM1[31:0],KM1[63:32],KM1[95:64],KM1[127:96]} ^ {KM2[127:96],KM2[95:6
4],KM2[63:32],KM2[31:0]};
  end
  else begin
    IK1 <= {KM1[95:64],KM1[63:32],KM1[31:0],KM1[127:96]} ^ {KM4[127:96],KM4[31:0]
,KM4[63:32],KM4[95:64]};
    IK2 <= {KM3[31:0],KM3[127:96],KM3[95:64],KM3[63:32]} ^ {KM4[31:0],KM4[127:96]
,KM4[95:64],KM4[63:32]};
    IK3 <= {KM1[31:0],KM1[127:96],KM1[95:64],KM1[63:32]} ^ {KM2[31:0],KM2[127:96]
,KM2[95:64],KM2[63:32]};
    IK4 <= {KM1[63:32],KM1[31:0],KM1[127:96],KM1[95:64]} ^ {KM4[127:96],KM4[95:6
4],KM4[63:32],KM4[31:0]};
    RK1 <= {KM3[31:0],KM3[63:32],KM3[95:64],KM3[127:96]} ^ {KM4[127:96],KM4[95:6
4],KM4[63:32],KM4[31:0]};
    RK2 <= {KM1[127:96],KM1[31:0],KM1[63:32],KM1[95:64]} ^ {KM2[95:64],KM2[63:32
],KM2[31:0],KM2[127:96]};
    RK3 <= {KM3[127:96],KM3[95:64],KM3[63:32],KM3[31:0]} ^ {KM4[63:32],KM4[31:0],
KM4[127:96],KM4[95:64]};
    RK4 <= {KM1[127:96],KM1[95:64],KM1[63:32],KM1[31:0]} ^ {KM4[31:0],KM4[63:32],
KM4[95:64],KM4[127:96]};
    RK5 <= {KM1[127:96],KM1[95:64],KM1[63:32],KM1[31:0]} ^ {KM2[63:32],KM2[31:0],
KM2[127:96],KM2[95:64]};
    RK6 <= {KM3[127:96],KM3[31:0],KM3[63:32],KM3[95:64]} ^ {KM4[95:64],KM4[63:32
],KM4[31:0],KM4[127:96]};
    RK7 <= {KM1[31:0],KM1[63:32],KM1[95:64],KM1[127:96]} ^ {KM2[127:96],KM2[95:6
4],KM2[63:32],KM2[31:0]};
  end
end
```

```
endmodule


/************************************************************
--  HyRAL128_ENC
--  Programed by K.I.Technology
--  2009.12.30
--  File Name: encrypt.v
************************************************************/

/*
----------------------------- 機能 -----------------------------
本モジュールは、128bit HyRAL暗号化機能を有します。
----------------------------------------------------------------

*/

/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   encrypt   ( CLK,    RST,    SDSW,  CSTART,
                IK1,    IK2,    IK3,
                IK4,    IK5,    IK6,
                RK1,    RK2,    RK3,
                RK4,    RK5,    RK6,
                RK7,    RK8,    RK9,
                PT,     CT,
                CEND,
                GREQ,    GSTART, GSEL,  GDATA,
                GEND,    GODATA,
                LFSTART, LFSEL, LFDATA, IK,
                LFEND,   LFODATA
              );

// Input/Output
input        CLK;                       // System Clock
input        RST;                       // System Reset
input        SDSW;
input        CSTART;

input        GEND;
input  [127:0] GODATA;

input        LFEND;
input  [127:0] LFODATA;

input  [127:0] IK1, IK2, IK3, IK4, IK5, IK6;
input  [127:0] RK1, RK2, RK3, RK4, RK5, RK6;
input  [127:0] RK7, RK8, RK9;

input  [127:0] PT;

output [127:0] CT;
```

```
output        CEND;

output        GREQ;
output        GSEL;
output [127:0] GDATA;
output        GSTART;

output        LFSEL, LFSTART;
output [127:0] LFDATA, IK;

reg    [127:0] LFDATA, GDATA, CT, IK;
reg    [  7:0] cstate;
reg            LFSEL, GSEL, GSTART, LFSTART;
reg            GREQ,  CEND;

parameter      CIDLE   = 8'h0,
               CSTATE1 = 8'h1,
               CSTATE2 = 8'h2,
               CSTATE3 = 8'h3,
               CSTATE4 = 8'h4,
               CSTATE5 = 8'h5,
               CSTATE6 = 8'h6,
               CSTATE7 = 8'h7,
               CSTATE8 = 8'h8;

always @(posedge CLK or posedge RST) begin
  if(RST) begin
    cstate  <= CIDLE;
    GSTART  <= 1'h0;
    LFSTART <= 1'h0;
    LFDATA  <= 128'h0;
    GDATA   <= 128'h0;
    GSEL    <= 1'h0;
    LFSEL   <= 1'h0;
    CT      <= 128'h0;
    IK      <= 128'h0;
    GREQ    <= 1'h0;
    CEND    <= 1'h0;
  end
  else
    case ( cstate )
    CIDLE   : if( CSTART ) begin
                cstate <= CSTATE1;
                GSTART <= 1'h1;
                GDATA  <= RK1 ^ PT;
                GSEL   <= 1'h0;          // G1
                GREQ   <= 1'h1;
                CEND   <= 1'h0;
              end
              else
                CEND   <= 1'h0;

    CSTATE1 : if( GEND ) begin
                cstate  <= CSTATE2;
```

```
          LFSTART <= 1'h1;
          LFDATA  <= RK2 ^ GODATA;
          LFSEL   <= 1'h1;          // F2
          IK      <= IK1;
        end
        else
          GSTART <= 1'h0;

CSTATE2 : if( LFEND ) begin
          cstate  <= CSTATE3;
          LFSTART <= 1'h1;
          LFDATA  <= RK3 ^ LFODATA;
          LFSEL   <= 1'h1;          // F2
          IK      <= IK2;
        end
        else
          LFSTART <= 1'h0;

CSTATE3 : if( LFEND ) begin
          cstate  <= CSTATE4;
          LFSTART <= 1'h1;
          LFDATA  <= RK4 ^ LFODATA;
          IK      <= IK3;
          if ( SDSW==1'h0 )
            LFSEL  <= 1'h0;         // F1
          else
            LFSEL  <= 1'h1;         // F2
        end
        else
          LFSTART <= 1'h0;

CSTATE4 : if( LFEND ) begin
          LFSTART <= 1'h1;
          LFDATA  <= RK5 ^ LFODATA;
          IK      <= IK4;
          LFSEL   <= 1'h0;          // F1
          if( SDSW )
            cstate <= CSTATE5;
          else
            cstate <= CSTATE7;
        end
        else
          LFSTART <= 1'h0;

CSTATE5 : if( LFEND ) begin
          cstate  <= CSTATE6;
          LFSTART <= 1'h1;
          LFDATA  <= RK6 ^ LFODATA;
          IK      <= IK5;
          LFSEL   <= 1'h0;          // F1
        end
        else
          LFSTART <= 1'h0;
```

```
    CSTATE6 : if( LFEND ) begin
              cstate  <= CSTATE7;
              LFSTART <= 1'h1;
              LFDATA  <= RK7 ^ LFODATA;
              IK      <= IK6;
              LFSEL   <= 1'h0;          // F1
            end
            else
              LFSTART <= 1'h0;

    CSTATE7 : if( LFEND ) begin
              cstate <= CSTATE8;
              GSTART <= 1'h1;
              GSEL   <= 1'h1;           // G2
              if( SDSW==1'h0 )
                GDATA  <= RK6 ^ LFODATA;
              else
                GDATA  <= RK8 ^ LFODATA;
            end
            else
              LFSTART <= 1'h0;

    CSTATE8 : if( GEND ) begin
              cstate <= CIDLE;
              GSEL   <= 1'h0;
              GREQ   <= 1'h0;
              CEND   <= 1'h1;
              if( SDSW==1'h0 )
                CT  <= RK7 ^ GODATA;
              else
                CT  <= RK9 ^ GODATA;
            end
            else
              GSTART <= 1'h0;
    endcase
end


endmodule


/*************************************************************
--  HyRAL128bit
--  Programed by K.I.Technology
--  2009.12.01
--  File Name: gfunc.v
*************************************************************/

/*
---------------------------- 機能 ----------------------------
本モジュールは、G function機能を有します。
--------------------------------------------------------------

*/
```

```
/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   gfunc ( CLK,     RST,
                 GSEL,    GSTART,   XDATA,
                 GEND,    YDATA,
                 FSTART,  FDATA, AT,       FREQ,
                 FEND,    FODATA
                );

// Input/Output
input         CLK;                  // System Clock
input         RST;                  // System Reset
input         GSEL;                  // G1,G2 SEL 0=G1,1=G2
input         GSTART;                  // START GFUNCTION
input  [127:0] XDATA;                     // 4Byte Input Data
input         FEND;
input  [ 31:0] FODATA;

output        GEND;                    // GFUNCTION END
output [127:0] YDATA;                     // 4Byte Output Data
output        FSTART;
output [ 31:0] FDATA;
output [  2:0] AT;
output        FREQ;

reg    [  7:0] gstate;
reg           GEND;
reg    [ 31:0] FDATA;
reg    [  2:0] AT;
reg           ENDFLG;
reg    [127:0] YDATA;                    // 4Byte Output Data
reg           FSTART;
reg    [ 31:0] LANE0, LANE1, LANE2, LANE3;
reg           FREQ;

parameter     GIDLE  = 8'h0,
              STAGE1 = 8'h1,
              STAGE2 = 8'h2,
              STAGE3 = 8'h3,
              STAGE4 = 8'h4;

always @(posedge CLK or posedge RST) begin
  if(RST) begin
    gstate <= GIDLE;
    FDATA  <= 32'h0;
    AT     <= 3'h0;
    FSTART <= 1'h0;
    LANE0  <= 32'h0;
    LANE1  <= 32'h0;
    LANE2  <= 32'h0;
    LANE3  <= 32'h0;
    ENDFLG <= 1'h0;
```

```
  GEND   <= 1'h0;
  YDATA  <= 128'h0;
  FREQ   <= 1'h0;
end
else
  case ( gstate )

  GIDLE:   if( GSTART ) begin
             gstate <= STAGE1;
             FSTART <= 1'h1;
             FDATA  <= XDATA[95:64] ^ XDATA[63:32] ^ XDATA[31: 0];
             FREQ   <= 1'h1;
             if ( GSEL==1'h0 )     // for G1
               AT     <= 3'h0;
             else                  // for G2
               AT     <= 3'h7;
           end

  STAGE1:  if( ENDFLG ) begin
             ENDFLG <= 1'h0;
             gstate <= STAGE2;
             FSTART <= 1'h1;
             FDATA  <= LANE1 ^ LANE2 ^ LANE3;
             if ( GSEL==1'h0 )     // for G1
               AT     <= 3'h1;
             else                  // for G2
               AT     <= 3'h6;
             end
           else if( FEND ) begin             // COMMON
             ENDFLG <= 1'h1;                 // Go to Next Stage FLAG
             LANE0  <= XDATA[95:64];
             LANE1  <= XDATA[63:32];
             LANE2  <= XDATA[31: 0];
             LANE3  <= XDATA[127:96] ^ FODATA;
           end
           else
             FSTART <= 1'h0;

  STAGE2:    if( ENDFLG ) begin
             ENDFLG <= 1'h0;
             gstate <= STAGE3;
             FSTART <= 1'h1;
             FDATA  <= LANE1 ^ LANE2 ^ LANE3;
             if ( GSEL==1'h0 )     // for G1
               AT     <= 3'h2;
             else                  // for G2
               AT     <= 3'h5;
           end
           else if( FEND ) begin             // COMMON
             ENDFLG <= 1'h1;                 // Go to Next Stage FLAG
             LANE0  <= LANE1;
             LANE1  <= LANE2;
             LANE2  <= LANE3;
             LANE3  <= LANE0 ^ FODATA;
```

```
          end
          else
            FSTART <= 1'h0;

    STAGE3:    if( ENDFLG ) begin
            ENDFLG <= 1'h0;
            gstate <= STAGE4;
            FSTART <= 1'h1;
            FDATA  <= LANE1 ^ LANE2 ^ LANE3;
            if ( GSEL==1'h0 )     // for G1
              AT     <= 3'h3;
            else                  // for G2
              AT     <= 3'h4;
          end
          else if( FEND ) begin              // COMMON
            ENDFLG <= 1'h1;                  // Go to Next Stage FLAG
            LANE0  <= LANE1;
            LANE1  <= LANE2;
            LANE2  <= LANE3;
            LANE3  <= LANE0 ^ FODATA;
          end
          else
            FSTART <= 1'h0;

    STAGE4:  if( ENDFLG ) begin
            ENDFLG <= 1'h0;
            GEND   <= 1'h0;
            gstate <= GIDLE;
            FREQ   <= 1'h0;
          end
          else if( FEND ) begin              // COMMON
            ENDFLG       <= 1'h1;            // Go to Next Stage FLAG
            GEND         <= 1'h1;
            YDATA[127:96] <= LANE1;
            YDATA[ 95:64] <= LANE2;
            YDATA[ 63:32] <= LANE3;
            YDATA[ 31: 0] <= LANE0 ^ FODATA;
//            LANE0  <= LANE1;
//            LANE1  <= LANE2;
//            LANE2  <= LANE3;
//            LANE3  <= LANE0 ^ FODATA;
          end
          else
            FSTART <= 1'h0;
    endcase
end



endmodule


/*************************************************************
--  HyRAL128bit
```

```
--  Programed by K.I.Technology
--  2009.12.01
--  File Name: lffunc.v
************************************************************/

/*
----------------------------- 機能 -----------------------------
本モジュールは、F function機能を有します。
------------------------------------------------------------------
*/

/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   lffunc( CLK,     RST,
                 LFSTART, FSEL,   XDATA,   IK,
                 LFEND,   YDATA,
                 FSTART,  FDATA,  AT,      FREQ,
                 FEND,    FODATA
               );

// Input/Output
input         CLK;                      // System Clock
input         RST;                      // System Reset
input         LFSTART;
input         FSEL;                     // F1 or F2 SEL 0=F1, 1=F2

input  [127:0] XDATA;
input  [127:0] IK;

input         FEND;
input  [ 31:0] FODATA;

output        LFEND;
output [127:0] YDATA;

output        FSTART;
output [ 31:0] FDATA;
output [  2:0] AT;
output        FREQ;

reg    [  7:0] fstate;
reg           LFEND;
reg    [ 31:0] FDATA;
reg    [ 31:0] LANE0, LANE1, LANE2, LANE3;
reg    [  2:0] AT;
reg           ENDFLG;
reg    [127:0] YDATA;                   // 4Byte Output Data
reg           FSTART;
reg           FREQ;

parameter     FIDLE   = 8'h0,
              FSTAGE1 = 8'h1,
              FSTAGE2 = 8'h2,
```

```
            FSTAGE3 = 8'h3,
            FSTAGE4 = 8'h4,
            FSTAGE5 = 8'h5,
            FSTAGE6 = 8'h6,
            FSTAGE7 = 8'h7,
            FSTAGE8 = 8'h8;

always @(posedge CLK or posedge RST) begin
  if(RST) begin
    fstate <= FIDLE;
    FDATA  <= 32'h0;
    AT     <= 3'h0;
    FSTART <= 1'h0;
    LANE0  <= 32'h0;
    LANE1  <= 32'h0;
    LANE2  <= 32'h0;
    LANE3  <= 32'h0;
    ENDFLG <= 1'h0;
    LFEND  <= 1'h0;
    YDATA  <= 128'h0;
    FREQ   <= 1'h0;
  end
  else
    case ( fstate )

    FIDLE:   if( LFSTART ) begin
               fstate <= FSTAGE1;
               FSTART <= 1'h1;
               FDATA  <= XDATA[31: 0];
               FREQ   <= 1'h1;
               if ( FSEL==1'h0 )     // for F1
                 AT     <= 3'h3;
               else                  // for F2
                 AT     <= 3'h4;
             end

    FSTAGE1: if( FEND ) begin
               fstate <= FSTAGE2;
               FSTART <= 1'h1;
               if ( FSEL==1'h0 ) begin     // for F1
                 FDATA  <= FODATA ^ IK[31:0] ^ XDATA[127:96];
                 AT     <= 3'h2;
               end
               else begin                  // for F2
                 FDATA  <= FODATA ^ IK[127:96] ^ XDATA[63:32];
                 AT     <= 3'h5;
               end
             end
             else
               FSTART <= 1'h0;


    FSTAGE2: if( ENDFLG ) begin
               ENDFLG <= 1'h0;
```

```
          fstate <= FSTAGE3;
          FSTART <= 1'h1;
          FDATA  <= LANE3;
          if ( FSEL==1'h0 )     // for F1
            AT     <= 3'h1;
          else                  // for F2
            AT     <= 3'h6;
        end
        else if( FEND ) begin                   // COMMON
          ENDFLG       <= 1'h1;                 // Go to Next Stage FLAG
          LANE0        <= XDATA[ 31: 0];
          LANE1        <= XDATA[127:96];
          LANE2        <= XDATA[ 95:64] ^ FODATA;
          LANE3        <= XDATA[ 63:32];
          YDATA[95:64] <= XDATA[ 95:64] ^ FODATA;
        end
        else
          FSTART <= 1'h0;

FSTAGE3: if( FEND ) begin
          fstate <= FSTAGE4;
          FSTART <= 1'h1;
          if ( FSEL==1'h0 ) begin    // for F1
            FDATA  <= FODATA ^ IK[63:32] ^ LANE0;
            AT     <= 3'h0;
          end
          else begin                 // for F2
            FDATA  <= FODATA ^ IK[95:64] ^ LANE2;
            AT     <= 3'h7;
          end
        end
        else
          FSTART <= 1'h0;


FSTAGE4: if( ENDFLG ) begin
          ENDFLG <= 1'h0;
          fstate <= FSTAGE5;
          FSTART <= 1'h1;
          FDATA  <= LANE3;
          if ( FSEL==1'h0 )     // for F1
            AT     <= 3'h2;
          else                  // for F2
            AT     <= 3'h5;
        end
        else if( FEND ) begin                   // COMMON
          ENDFLG       <= 1'h1;                 // Go to Next Stage FLAG
          LANE0        <= LANE3;
          LANE1        <= LANE0;
          LANE2        <= LANE1 ^ FODATA;
          LANE3        <= LANE2;
          YDATA[127:96] <= LANE1 ^ FODATA;
        end
        else
```

```
        FSTART <= 1'h0;

FSTAGE5: if( FEND ) begin
        fstate <= FSTAGE6;
        FSTART <= 1'h1;
        if ( FSEL==1'h0 ) begin    // for F1
          FDATA  <= FODATA ^ IK[95:64] ^ LANE0;
          AT     <= 3'h3;
        end
        else begin                 // for F2
          FDATA  <= FODATA ^ IK[63:32] ^ LANE2;
          AT     <= 3'h4;
        end
      end
      else
        FSTART <= 1'h0;


FSTAGE6: if( ENDFLG ) begin
        ENDFLG <= 1'h0;
        fstate <= FSTAGE7;
        FSTART <= 1'h1;
        FDATA  <= LANE3;
        if ( FSEL==1'h0 )     // for F1
          AT     <= 3'h0;
        else                  // for F2
          AT     <= 3'h7;
      end
      else if( FEND ) begin                  // COMMON
        ENDFLG       <= 1'h1;                // Go to Next Stage FLAG
        LANE0        <= LANE3;
        LANE1        <= LANE0;
        LANE2        <= LANE1 ^ FODATA;
        LANE3        <= LANE2;
        YDATA[ 31: 0] <= LANE1 ^ FODATA;
      end
      else
        FSTART <= 1'h0;

FSTAGE7: if( FEND ) begin
        fstate <= FSTAGE8;
        FSTART <= 1'h1;
        if ( FSEL==1'h0 ) begin    // for F1
          FDATA  <= FODATA ^ IK[127:96] ^ LANE0;
          AT     <= 3'h1;
        end
        else begin                 // for F2
          FDATA  <= FODATA ^ IK[31:0] ^ LANE2;
          AT     <= 3'h6;
        end
      end
      else
        FSTART <= 1'h0;
```

```
    FSTAGE8: if( ENDFLG ) begin
               ENDFLG <= 1'h0;
               LFEND  <= 1'h0;
               FREQ   <= 1'h0;
               fstate <= FIDLE;
             end
             else if( FEND ) begin                  // COMMON
               ENDFLG       <= 1'h1;               // Go to Next Stage FLAG
               LFEND        <= 1'h1;
               LANE0        <= LANE3;
               LANE1        <= LANE0;
               LANE2        <= LANE1 ^ FODATA;
               LANE3        <= LANE2;
               YDATA[ 63:32] <= LANE1 ^ FODATA;
             end
             else
               FSTART <= 1'h0;
    endcase
end


endmodule


/*************************************************************
--  HyRAL128bit
--  Programed by K.I.Technology
--  2009.12.01
--  File Name: decrypt.v
*************************************************************/

/*
---------------------------- 機能 ----------------------------
本モジュールは、128bit HyRAL復号化機能を有します。
--------------------------------------------------------------

*/

/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   decrypt   ( CLK,    RST,    SDSW,   CSTART,
                     IK1,    IK2,    IK3,
                     IK4,    IK5,    IK6,
                     RK1,    RK2,    RK3,
                     RK4,    RK5,    RK6,
                     RK7,    RK8,    RK9,
                     PT,     CT,
                     CEND,
                     GSTART,   GSEL,  GDATA,
                     GEND,    GODATA,
                     LFSTART, LFSEL,  LFDATA, IK,
                     LFEND,   LFODATA
```

```
                );

// Input/Output
input         CLK;                      // System Clock
input         RST;                      // System Reset
input         SDSW;
input         CSTART;

input         GEND;
input  [127:0] GODATA;

input         LFEND;
input  [127:0] LFODATA;

input  [127:0] IK1, IK2, IK3, IK4, IK5, IK6;
input  [127:0] RK1, RK2, RK3, RK4, RK5, RK6;
input  [127:0] RK7, RK8, RK9;

input  [127:0] PT;

output [127:0] CT;
output         CEND;

output         GSEL;
output [127:0] GDATA;
output         GSTART;

output         LFSEL, LFSTART;
output [127:0] LFDATA, IK;

reg    [127:0] LFDATA, GDATA, CT, IK;
reg    [  7:0] cstate;
reg            LFSEL, GSEL, GSTART, LFSTART;
reg            CEND;

parameter      CIDLE   = 8'h0,
               CSTATE1 = 8'h1,
               CSTATE2 = 8'h2,
               CSTATE3 = 8'h3,
               CSTATE4 = 8'h4,
               CSTATE5 = 8'h5,
               CSTATE6 = 8'h6,
               CSTATE7 = 8'h7,
               CSTATE8 = 8'h8;

always @(posedge CLK or posedge RST) begin
  if(RST) begin
    cstate  <= CIDLE;
    GSTART  <= 1'h0;
    LFSTART <= 1'h0;
    LFDATA  <= 128'h0;
    GDATA   <= 128'h0;
    GSEL    <= 1'h0;
    LFSEL   <= 1'h0;
```

```
  CT      <= 128'h0;
  IK      <= 128'h0;
  CEND    <= 1'h0;
end
else
  case ( cstate )
  CIDLE   : if( CSTART ) begin
            cstate <= CSTATE1;
            GSTART <= 1'h1;
            GSEL   <= 1'h1;          // G2
            CEND   <= 1'h0;
            if(SDSW == 1'h0)
              GDATA  <= RK7 ^ PT;
            else
              GDATA  <= RK9 ^ PT;
          end
          else
            CEND   <= 1'h0;

  CSTATE1 : if( GEND ) begin
            cstate  <= CSTATE2;
            LFSTART <= 1'h1;
            LFSEL   <= 1'h0;          // F1
            if(SDSW == 1'h0) begin
              LFDATA  <= RK6 ^ GODATA;
              IK      <= IK4;
            end
            else begin
              LFDATA  <= RK8 ^ GODATA;
              IK      <= IK6;
            end
          end
          else
            GSTART <= 1'h0;

  CSTATE2 : if( LFEND ) begin
            cstate  <= CSTATE3;
            LFSTART <= 1'h1;
            LFSEL   <= 1'h0;          // F1
            if(SDSW == 1'h0) begin
              LFDATA  <= RK5 ^ LFODATA;
              IK      <= IK3;
            end
            else begin
              LFDATA  <= RK7 ^ LFODATA;
              IK      <= IK5;
            end
          end
          else
            LFSTART <= 1'h0;

  CSTATE3 : if( LFEND ) begin
            cstate  <= CSTATE4;
            LFSTART <= 1'h1;
```

```
          if ( SDSW==1'h0 ) begin
            LFSEL  <= 1'h1;        // F2
            LFDATA  <= RK4 ^ LFODATA;
            IK      <= IK2;
          end
          else begin
            LFSEL  <= 1'h0;        // F1
            LFDATA  <= RK6 ^ LFODATA;
            IK      <= IK4;
          end
        end
        else
          LFSTART <= 1'h0;

CSTATE4 : if( LFEND ) begin
          LFSTART <= 1'h1;
          LFSEL   <= 1'h1;          // F2
          if( SDSW ) begin
            cstate <= CSTATE5;
            LFDATA  <= RK5 ^ LFODATA;
            IK      <= IK3;
          end
          else begin
            cstate <= CSTATE7;
            LFDATA  <= RK3 ^ LFODATA;
            IK      <= IK1;
          end
        end
        else
          LFSTART <= 1'h0;

CSTATE5 : if( LFEND ) begin
          cstate  <= CSTATE6;
          LFSTART <= 1'h1;
          LFDATA  <= RK4 ^ LFODATA;
          IK      <= IK2;
          LFSEL   <= 1'h1;          // F2
        end
        else
          LFSTART <= 1'h0;

CSTATE6 : if( LFEND ) begin
          cstate  <= CSTATE7;
          LFSTART <= 1'h1;
          LFDATA  <= RK3 ^ LFODATA;
          IK      <= IK1;
          LFSEL   <= 1'h1;          // F2
        end
        else
          LFSTART <= 1'h0;

CSTATE7 : if( LFEND ) begin
          cstate <= CSTATE8;
          GSTART <= 1'h1;
```

```
            GSEL   <= 1'h0;          // G1
            GDATA  <= RK2 ^ LFODATA;
          end
          else
            LFSTART <= 1'h0;

    CSTATE8 : if( GEND ) begin
            cstate <= CIDLE;
            GSEL   <= 1'h0;
            CEND   <= 1'h1;
            CT     <= RK1 ^ GODATA;
          end
          else
            GSTART <= 1'h0;
    endcase
end


endmodule


/*************************************************************
--  HyRAL128bit
--  Programed by K.I.Technology
--  2009.12.28
--  File Name: gifunc.v
*************************************************************/

/*
---------------------------- 構成 -----------------------------
HyRAL128_main--+-- intface
          +-- keygen
          +-- bkeygen
          +-- skeygen
          +-- encrypt
          +-- gfunc
          +-- lffunc
          +-- decrypt
          +-- gifunc
          +-- lfifunc
          +-- ffunc    -- sbox_rom
-------------------------------------------------------------

---------------------------- 機能 -----------------------------
本モジュールは、G inverse function機能を有します。
-------------------------------------------------------------
*/

/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   gifunc ( CLK,    RST,
            GSEL,   GSTART,  XDATA,
            GEND,   YDATA,
```

```
              FSTART,  FDATA,    AT,      FREQ,
              FEND,    FODATA
             );

// Input/Output
input         CLK;                // System Clock
input         RST;                // System Reset
input         GSEL;               // G1,G2 SEL 0=G1,1=G2
input         GSTART;               // START GFUNCTION
input  [127:0] XDATA;                // 4Byte Input Data
input         FEND;
input  [ 31:0] FODATA;

output        GEND;                 // GFUNCTION END
output [127:0] YDATA;                // 4Byte Output Data
output        FSTART;
output [ 31:0] FDATA;
output [  2:0] AT;
output        FREQ;

reg    [  7:0] gstate;
reg           GEND;
reg    [ 31:0] FDATA;
reg    [  2:0] AT;
reg           ENDFLG;
reg    [127:0] YDATA;                // 4Byte Output Data
reg           FSTART;
reg    [ 31:0] LANE0, LANE1, LANE2, LANE3;
reg           FREQ;

parameter     GIDLE  = 8'h0,
              STAGE1 = 8'h1,
              STAGE2 = 8'h2,
              STAGE3 = 8'h3,
              STAGE4 = 8'h4;

always @(posedge CLK or posedge RST) begin
  if(RST) begin
    gstate <= GIDLE;
    FDATA  <= 32'h0;
    AT     <= 3'h0;
    FSTART <= 1'h0;
    LANE0  <= 32'h0;
    LANE1  <= 32'h0;
    LANE2  <= 32'h0;
    LANE3  <= 32'h0;
    ENDFLG <= 1'h0;
    GEND   <= 1'h0;
    YDATA  <= 128'h0;
    FREQ   <= 1'h0;
  end
  else
    case ( gstate )
```

```
GIDLE:   if( GSTART ) begin
           gstate <= STAGE1;
           FSTART <= 1'h1;
           FDATA  <= XDATA[95:64] ^ XDATA[63:32] ^ XDATA[127:96];
           FREQ   <= 1'h1;
           if ( GSEL==1'h0 )     // for G1
             AT     <= 3'h3;
           else                  // for G2
             AT     <= 3'h4;
         end

STAGE1:  if( ENDFLG ) begin
           ENDFLG <= 1'h0;
           gstate <= STAGE2;
           FSTART <= 1'h1;
           FDATA  <= LANE1 ^ LANE2 ^ LANE3;
           if ( GSEL==1'h0 )     // for G1
             AT     <= 3'h2;
           else                  // for G2
             AT     <= 3'h5;
           end
         else if( FEND ) begin             // COMMON
           ENDFLG <= 1'h1;                  // Go to Next Stage FLAG
           LANE0  <= XDATA[ 63:32];
           LANE1  <= XDATA[ 95:64];
           LANE2  <= XDATA[127:96];
           LANE3  <= XDATA[ 31: 0] ^ FODATA;
         end
         else
           FSTART <= 1'h0;

STAGE2:    if( ENDFLG ) begin
           ENDFLG <= 1'h0;
           gstate <= STAGE3;
           FSTART <= 1'h1;
           FDATA  <= LANE1 ^ LANE2 ^ LANE3;
           if ( GSEL==1'h0 )     // for G1
             AT     <= 3'h1;
           else                  // for G2
             AT     <= 3'h6;
         end
         else if( FEND ) begin             // COMMON
           ENDFLG <= 1'h1;                  // Go to Next Stage FLAG
           LANE0  <= LANE1;
           LANE1  <= LANE2;
           LANE2  <= LANE3;
           LANE3  <= LANE0 ^ FODATA;
         end
         else
           FSTART <= 1'h0;

STAGE3:    if( ENDFLG ) begin
           ENDFLG <= 1'h0;
           gstate <= STAGE4;
```

```
            FSTART <= 1'h1;
            FDATA  <= LANE1 ^ LANE2 ^ LANE3;
            if ( GSEL==1'h0 )     // for G1
              AT     <= 3'h0;
            else                  // for G2
              AT     <= 3'h7;
          end
          else if( FEND ) begin             // COMMON
            ENDFLG <= 1'h1;                  // Go to Next Stage FLAG
            LANE0  <= LANE1;
            LANE1  <= LANE2;
            LANE2  <= LANE3;
            LANE3  <= LANE0 ^ FODATA;
          end
          else
            FSTART <= 1'h0;

    STAGE4:  if( ENDFLG ) begin
            ENDFLG <= 1'h0;
            GEND   <= 1'h0;
            gstate <= GIDLE;
            FREQ   <= 1'h0;
          end
          else if( FEND ) begin             // COMMON
            ENDFLG       <= 1'h1;          // Go to Next Stage FLAG
            GEND         <= 1'h1;
            YDATA[ 31: 0] <= LANE1;
            YDATA[ 63:32] <= LANE2;
            YDATA[ 95:64] <= LANE3;
            YDATA[127:96] <= LANE0 ^ FODATA;
//            LANE0  <= LANE1;
//            LANE1  <= LANE2;
//            LANE2  <= LANE3;
//            LANE3  <= LANE0 ^ FODATA;
          end
          else
            FSTART <= 1'h0;
    endcase
end



endmodule


/*************************************************************
--  HyRAL128bit
--  Programed by K.I.Technology
--  2009.12.01
--  File Name: lfifunc.v
*************************************************************/

/*
---------------------------- 機能 ----------------------------
```

```
本モジュールは、F inverse function機能を有します。
-----------------------------------------------------------------

*/

/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   lfifunc( CLK,     RST,
             LFSTART, FSEL,   XDATA,   IK,
             LFEND,   YDATA,
             FSTART,  FDATA,  AT,      FREQ,
             FEND,    FODATA
           );

// Input/Output
input         CLK;                    // System Clock
input         RST;                    // System Reset
input         LFSTART;
input         FSEL;                   // F1 or F2 SEL 0=F1, 1=F2

input  [127:0] XDATA;
input  [127:0] IK;

input         FEND;
input  [ 31:0] FODATA;

output        LFEND;
output [127:0] YDATA;

output        FSTART;
output [ 31:0] FDATA;
output [  2:0] AT;
output        FREQ;

reg    [  7:0] fstate;
reg           LFEND;
reg    [ 31:0] FDATA;
reg    [ 31:0] LANE0, LANE1, LANE2, LANE3;
reg    [  2:0] AT;
reg           ENDFLG;
reg    [127:0] YDATA;                 // 4Byte Output Data
reg           FSTART;
reg           FREQ;

parameter     FIDLE   = 8'h0,
              FSTAGE1 = 8'h1,
              FSTAGE2 = 8'h2,
              FSTAGE3 = 8'h3,
              FSTAGE4 = 8'h4,
              FSTAGE5 = 8'h5,
              FSTAGE6 = 8'h6,
              FSTAGE7 = 8'h7,
              FSTAGE8 = 8'h8;
```

```
always @(posedge CLK or posedge RST) begin
  if(RST) begin
    fstate <= FIDLE;
    FDATA  <= 32'h0;
    AT     <= 3'h0;
    FSTART <= 1'h0;
    LANE0  <= 32'h0;
    LANE1  <= 32'h0;
    LANE2  <= 32'h0;
    LANE3  <= 32'h0;
    ENDFLG <= 1'h0;
    LFEND  <= 1'h0;
    YDATA  <= 128'h0;
    FREQ   <= 1'h0;
  end
  else
    case ( fstate )

    FIDLE:   if( LFSTART ) begin
               fstate <= FSTAGE1;
               FSTART <= 1'h1;
               FDATA  <= XDATA[127:96];
               FREQ   <= 1'h1;
               if ( FSEL==1'h0 )     // for F1
                 AT     <= 3'h0;
               else                  // for F2
                 AT     <= 3'h7;
             end

    FSTAGE1: if( FEND ) begin
               fstate <= FSTAGE2;
               FSTART <= 1'h1;
               if ( FSEL==1'h0 ) begin     // for F1
                 FDATA  <= FODATA ^ IK[127:96] ^ XDATA[95:64];
                 AT     <= 3'h1;
               end
               else begin                  // for F2
                 FDATA  <= FODATA ^ IK[31:0] ^ XDATA[31:0];
                 AT     <= 3'h6;
               end
             end
             else
               FSTART <= 1'h0;


    FSTAGE2: if( ENDFLG ) begin
               ENDFLG <= 1'h0;
               fstate <= FSTAGE3;
               FSTART <= 1'h1;
               FDATA  <= LANE3;
               if ( FSEL==1'h0 )     // for F1
                 AT     <= 3'h2;
               else                  // for F2
```

```
          AT     <= 3'h5;
        end
        else if( FEND ) begin                  // COMMON
          ENDFLG      <= 1'h1;                 // Go to Next Stage FLAG
          LANE0       <= XDATA[127:96];
          LANE1       <= XDATA[ 31: 0];
          LANE2       <= XDATA[ 63:32] ^ FODATA;
          LANE3       <= XDATA[ 95:64];
          YDATA[63:32] <= XDATA[ 63:32] ^ FODATA;
        end
        else
          FSTART <= 1'h0;

FSTAGE3: if( FEND ) begin
          fstate <= FSTAGE4;
          FSTART <= 1'h1;
          if ( FSEL==1'h0 ) begin    // for F1
            FDATA  <= FODATA ^ IK[95:64] ^ LANE2;
            AT     <= 3'h3;
          end
          else begin                 // for F2
            FDATA  <= FODATA ^ IK[63:32] ^ LANE0;
            AT     <= 3'h4;
          end
        end
        else
          FSTART <= 1'h0;


FSTAGE4: if( ENDFLG ) begin
          ENDFLG <= 1'h0;
          fstate <= FSTAGE5;
          FSTART <= 1'h1;
          FDATA  <= LANE3;
          if ( FSEL==1'h0 )     // for F1
            AT     <= 3'h1;
          else                  // for F2
            AT     <= 3'h6;
        end
        else if( FEND ) begin                  // COMMON
          ENDFLG      <= 1'h1;                 // Go to Next Stage FLAG
          LANE0       <= LANE3;
          LANE1       <= LANE0;
          LANE2       <= LANE1 ^ FODATA;
          LANE3       <= LANE2;
          YDATA[ 31: 0] <= LANE1 ^ FODATA;
        end
        else
          FSTART <= 1'h0;

FSTAGE5: if( FEND ) begin
          fstate <= FSTAGE6;
          FSTART <= 1'h1;
          if ( FSEL==1'h0 ) begin    // for F1
```

```
            FDATA  <= FODATA ^ IK[63:32] ^ LANE2;
            AT     <= 3'h0;
          end
          else begin                // for F2
            FDATA  <= FODATA ^ IK[95:64] ^ LANE0;
            AT     <= 3'h7;
          end
        end
        else
          FSTART <= 1'h0;


FSTAGE6: if( ENDFLG ) begin
          ENDFLG <= 1'h0;
          fstate <= FSTAGE7;
          FSTART <= 1'h1;
          FDATA  <= LANE3;
          if ( FSEL==1'h0 )     // for F1
            AT     <= 3'h3;
          else                  // for F2
            AT     <= 3'h4;
        end
        else if( FEND ) begin                   // COMMON
          ENDFLG       <= 1'h1;                 // Go to Next Stage FLAG
          LANE0        <= LANE3;
          LANE1        <= LANE0;
          LANE2        <= LANE1 ^ FODATA;
          LANE3        <= LANE2;
          YDATA[127:96] <= LANE1 ^ FODATA;
        end
        else
          FSTART <= 1'h0;

FSTAGE7: if( FEND ) begin
          fstate <= FSTAGE8;
          FSTART <= 1'h1;
          if ( FSEL==1'h0 ) begin    // for F1
            FDATA  <= FODATA ^ IK[31:0] ^ LANE2;
            AT     <= 3'h2;
          end
          else begin                // for F2
            FDATA  <= FODATA ^ IK[127:96] ^ LANE0;
            AT     <= 3'h5;
          end
        end
        else
          FSTART <= 1'h0;


FSTAGE8: if( ENDFLG ) begin
          ENDFLG <= 1'h0;
          LFEND  <= 1'h0;
          FREQ   <= 1'h0;
          fstate <= FIDLE;
```

```
          end
          else if( FEND ) begin                  // COMMON
            ENDFLG       <= 1'h1;               // Go to Next Stage FLAG
            LFEND        <= 1'h1;
            LANE0        <= LANE3;
            LANE1        <= LANE0;
            LANE2        <= LANE1 ^ FODATA;
            LANE3        <= LANE2;
            YDATA[ 95:64] <= LANE1 ^ FODATA;
          end
          else
            FSTART <= 1'h0;
    endcase
end


endmodule


/************************************************************
--  HyRAL128bit
--  Programed by K.I.Technology
--  2009.12.30
--  File Name: ffunc.v
************************************************************/

/*
---------------------------- 機能 ----------------------------
本モジュールは、f function機能を有します。
--------------------------------------------------------------

*/

/* シュミレーション時の分解能記述 */
`timescale 100ps/100ps

module   ffunc ( CLK,     RST,
            FSTART,  XDATA,   KEY,     AT,
            CST1,    CST2,    CST3,    CST4,
            FEND,    YDATA
          );

// Input/Output
input        CLK;                  // System Clock
input        RST;                  // System Reset
input        FSTART;                 // f function START

input  [31:0] XDATA;                   // 4Byte Input Data
input  [31:0] KEY;                   // 4Byte Input Key
input  [ 2:0] AT;                    // Transrate number
input  [ 7:0] CST1, CST2, CST3, CST4;

output       FEND;                  // f function END
output [31:0] YDATA;                   // 4Byte Output Data
```

```
wire   [31:0] XKEOR;
reg    [31:0] XKEORD;
wire   [31:0] SBOXA;
wire   [ 9:0] A11, A12, A13, A14;
wire   [ 9:0] A21, A22, A23, A24;
wire   [ 9:0] A31, A32, A33, A34;
wire   [ 9:0] A41, A42, A43, A44;
reg    [ 7:0] A11A, A12A, A13A, A14A;
reg    [ 7:0] A21A, A22A, A23A, A24A;
reg    [ 7:0] A31A, A32A, A33A, A34A;
reg    [ 7:0] A41A, A42A, A43A, A44A;
wire   [ 7:0] Y0, Y1, Y2, Y3;
reg    [ 3:0] PIPELN;
reg           fEN;

parameter     PIPELINE = 4'h3;         // PIPELINE for f function enzan

// FSTART, FEND Generate
assign FEND = fEN & (PIPELN==PIPELINE);
always @(posedge CLK or posedge RST)begin
  if(RST)
    fEN <= 1'h0;
  else if( FEND==1'h1 )
    fEN <= 1'h0;
  else if( FSTART )
    fEN <= 1'h1;
end

always @(posedge CLK or posedge RST)begin
  if(RST)
    PIPELN <= 4'h0;
  else if( FEND==1'h1 )
    PIPELN <= 4'h0;
  else if( fEN==1'h1 )
    PIPELN <= PIPELN + 1;
end

assign XKEOR = XDATA ^ KEY;

always @(posedge CLK or posedge RST)begin
  if(RST)
    XKEORD <= 32'h0;
  else
    case(AT)
      3'h0 : XKEORD <= XKEOR;                                          // f1
      3'h1 : XKEORD <= {XKEOR[23:16], XKEOR[15: 8], XKEOR[ 7: 0], XKEOR[31:24]};  //
f2
      3'h2 : XKEORD <= {XKEOR[15: 8], XKEOR[ 7: 0], XKEOR[31:24], XKEOR[23:16]};  //
f3
      3'h3 : XKEORD <= {XKEOR[ 7: 0], XKEOR[31:24], XKEOR[23:16], XKEOR[15: 8]};  //
f4
      3'h4 : XKEORD <= {XKEOR[ 7: 0], XKEOR[15: 8], XKEOR[23:16], XKEOR[31:24]};  //
f5
```

```
      3'h5 : XKEORD <= {XKEOR[15: 8], XKEOR[23:16], XKEOR[31:24], XKEOR[ 7: 0]};  //
f6
      3'h6 : XKEORD <= {XKEOR[23:16], XKEOR[31:24], XKEOR[ 7: 0], XKEOR[15: 8]};  //
f7
      3'h7 : XKEORD <= {XKEOR[31:24], XKEOR[ 7: 0], XKEOR[15: 8], XKEOR[23:16]};  //
f8
    endcase
end

// SBOX
// SBOX is used LUT.
sbox_rom A0( .clka( CLK ),
          .addra( XKEORD[31:24] ),
          .douta( SBOXA[31:24] )
         );

sbox_rom A1( .clka( CLK ),
          .addra( XKEORD[23:16] ),
          .douta( SBOXA[23:16] )
         );

sbox_rom A2( .clka( CLK ),
          .addra( XKEORD[15: 8] ),
          .douta( SBOXA[15: 8] )
         );

sbox_rom A3( .clka( CLK ),
          .addra( XKEORD[ 7: 0] ),
          .douta( SBOXA[ 7: 0] )
         );

// MDS
// This MDS calculator is fixed coefficient case only.

// YDATA0
assign A11 = {2'h0, SBOXA[31:24]} ^ {1'h0, SBOXA[31:24], 1'h0};                    // 0x3
assign A12 = {2'h0, SBOXA[23:16]} ^ {1'h0, SBOXA[23:16], 1'h0};                    // 0x3
assign A13 =                  {1'h0, SBOXA[15: 8], 1'h0};                  // 0x2
assign A14 = {2'h0, SBOXA[ 7: 0]} ;                                      // 0x1

always @(posedge CLK or posedge RST)begin
  if(RST)
    A11A <= 8'h0;
  else
    case(A11[9:8])
      2'h0 : A11A <= A11[7:0];
      2'h1 : A11A <= A11[7:0] ^ 8'h1B;
      2'h2 : A11A <= A11[7:0] ^ 8'h36;
      2'h3 : A11A <= A11[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
    A12A <= 8'h0;
```

```
  else
    case(A12[9:8])
      2'h0 : A12A <= A12[7:0];
      2'h1 : A12A <= A12[7:0] ^ 8'h1B;
      2'h2 : A12A <= A12[7:0] ^ 8'h36;
      2'h3 : A12A <= A12[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
    A13A <= 8'h0;
  else
    case(A13[9:8])
      2'h0 : A13A <= A13[7:0];
      2'h1 : A13A <= A13[7:0] ^ 8'h1B;
      2'h2 : A13A <= A13[7:0] ^ 8'h36;
      2'h3 : A13A <= A13[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
    A14A <= 8'h0;
  else
    case(A14[9:8])
      2'h0 : A14A <= A14[7:0];
      2'h1 : A14A <= A14[7:0] ^ 8'h1B;
      2'h2 : A14A <= A14[7:0] ^ 8'h36;
      2'h3 : A14A <= A14[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
assign Y0 = A11A ^ A12A ^ A13A ^ A14A ^ CST1;  // CST1 is expected 0x11.
assign YDATA[31:24] = Y0;


// YDATA1
assign A21 = {2'h0, SBOXA[31:24]};                                  // 0x1
assign A22 =                {1'h0, SBOXA[23:16], 1'h0};               // 0x2
assign A23 =                {1'h0, SBOXA[15: 8], 1'h0};               // 0x2
assign A24 =                {1'h0, SBOXA[ 7: 0], 1'h0};               // 0x2

always @(posedge CLK or posedge RST)begin
  if(RST)
    A21A <= 8'h0;
  else
    case(A21[9:8])
      2'h0 : A21A <= A21[7:0];
      2'h1 : A21A <= A21[7:0] ^ 8'h1B;
      2'h2 : A21A <= A21[7:0] ^ 8'h36;
      2'h3 : A21A <= A21[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
    A22A <= 8'h0;
```

```
  else
    case(A22[9:8])
      2'h0 : A22A <= A22[7:0];
      2'h1 : A22A <= A22[7:0] ^ 8'h1B;
      2'h2 : A22A <= A22[7:0] ^ 8'h36;
      2'h3 : A22A <= A22[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
    A23A <= 8'h0;
  else
    case(A23[9:8])
      2'h0 : A23A <= A23[7:0];
      2'h1 : A23A <= A23[7:0] ^ 8'h1B;
      2'h2 : A23A <= A23[7:0] ^ 8'h36;
      2'h3 : A23A <= A23[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
    A24A <= 8'h0;
  else
    case(A24[9:8])
      2'h0 : A24A <= A24[7:0];
      2'h1 : A24A <= A24[7:0] ^ 8'h1B;
      2'h2 : A24A <= A24[7:0] ^ 8'h36;
      2'h3 : A24A <= A24[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
assign Y1 = A21A ^ A22A ^ A23A ^ A24A ^ CST2;  // CST2 is expected 0x22.
assign YDATA[23:16] = Y1;


// YDATA2
assign A31 = {2'h0, SBOXA[31:24]} ^ {1'h0, SBOXA[31:24], 1'h0} ^ {SBOXA[31:24], 2'h0};
  // 0x7
assign A32 = {2'h0, SBOXA[23:16]} ^ {1'h0, SBOXA[23:16], 1'h0};                    // 0x3
assign A33 = {2'h0, SBOXA[15: 8]};                                          // 0x1
assign A34 =                    {1'h0, SBOXA[ 7: 0], 1'h0};                  // 0x2

always @(posedge CLK or posedge RST)begin
  if(RST)
    A31A <= 8'h0;
  else
    case(A31[9:8])
      2'h0 : A31A <= A31[7:0];
      2'h1 : A31A <= A31[7:0] ^ 8'h1B;
      2'h2 : A31A <= A31[7:0] ^ 8'h36;
      2'h3 : A31A <= A31[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
```

```
    A32A <= 8'h0;
  else
    case(A32[9:8])
      2'h0 : A32A <= A32[7:0];
      2'h1 : A32A <= A32[7:0] ^ 8'h1B;
      2'h2 : A32A <= A32[7:0] ^ 8'h36;
      2'h3 : A32A <= A32[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
    A33A <= 8'h0;
  else
    case(A33[9:8])
      2'h0 : A33A <= A33[7:0];
      2'h1 : A33A <= A33[7:0] ^ 8'h1B;
      2'h2 : A33A <= A33[7:0] ^ 8'h36;
      2'h3 : A33A <= A33[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
    A34A <= 8'h0;
  else
    case(A34[9:8])
      2'h0 : A34A <= A34[7:0];
      2'h1 : A34A <= A34[7:0] ^ 8'h1B;
      2'h2 : A34A <= A34[7:0] ^ 8'h36;
      2'h3 : A34A <= A34[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
assign Y2 = A31A ^ A32A ^ A33A ^ A34A ^ CST3;  // CST3 is expected 0x44.
assign YDATA[15: 8] = Y2;

// YDATA3
assign A41 = {2'h0, SBOXA[31:24]} ^ {1'h0, SBOXA[31:24], 1'h0} ^ {SBOXA[31:24], 2'h0};
 // 0x7
assign A42 =                                          {SBOXA[23:16], 2'h0};  // 0x4
assign A43 = {2'h0, SBOXA[15: 8]} ^                           {SBOXA[15: 8], 2'h0};  // 0x5
assign A44 = {2'h0, SBOXA[ 7: 0]} ^ {1'h0, SBOXA[ 7: 0], 1'h0};                      // 0x3

always @(posedge CLK or posedge RST)begin
  if(RST)
    A41A <= 8'h0;
  else
    case(A41[9:8])
      2'h0 : A41A <= A41[7:0];
      2'h1 : A41A <= A41[7:0] ^ 8'h1B;
      2'h2 : A41A <= A41[7:0] ^ 8'h36;
      2'h3 : A41A <= A41[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
```

```
    A42A <= 8'h0;
  else
    case(A42[9:8])
      2'h0 : A42A <= A42[7:0];
      2'h1 : A42A <= A42[7:0] ^ 8'h1B;
      2'h2 : A42A <= A42[7:0] ^ 8'h36;
      2'h3 : A42A <= A42[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
    A43A <= 8'h0;
  else
    case(A43[9:8])
      2'h0 : A43A <= A43[7:0];
      2'h1 : A43A <= A43[7:0] ^ 8'h1B;
      2'h2 : A43A <= A43[7:0] ^ 8'h36;
      2'h3 : A43A <= A43[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
always @(posedge CLK or posedge RST)begin
  if(RST)
    A44A <= 8'h0;
  else
    case(A44[9:8])
      2'h0 : A44A <= A44[7:0];
      2'h1 : A44A <= A44[7:0] ^ 8'h1B;
      2'h2 : A44A <= A44[7:0] ^ 8'h36;
      2'h3 : A44A <= A44[7:0] ^ 8'h36 ^ 8'h1B;
    endcase
end
assign Y3 = A41A ^ A42A ^ A43A ^ A44A ^ CST4;  // CST4 is expected 0x88.
assign YDATA[ 7: 0] = Y3;

endmodule


/*********************************************************************************
*     This file is owned and controlled by Xilinx and must be used              *
*     solely for design, simulation, implementation and creation of             *
*     design files limited to Xilinx devices or technologies. Use               *
*     with non-Xilinx devices or technologies is expressly prohibited           *
*     and immediately terminates your license.                                  *
*                                                                               *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"             *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR                   *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION           *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION               *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS                 *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,                   *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE          *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY                  *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                   *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR            *
```

```
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF        *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        *
*     FOR A PARTICULAR PURPOSE.                                              *
*                                                                            *
*     Xilinx products are not intended for use in life support               *
*     appliances, devices, or systems. Use in such applications are          *
*     expressly prohibited.                                                  *
*                                                                            *
*     (c) Copyright 1995-2007 Xilinx, Inc.                                   *
*     All rights reserved.                                                   *
*****************************************************************************/
// The synthesis directives "translate_off/translate_on" specified below are
// supported by Xilinx, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file sbox_rom.v when simulating
// the core, sbox_rom. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps

module sbox_rom(
  clka,
  addra,
  douta);


input clka;
input [7 : 0] addra;
output [7 : 0] douta;

// synthesis translate_off

     BLK_MEM_GEN_V2_7 #(
     .C_ADDRA_WIDTH(8),
     .C_ADDRB_WIDTH(8),
     .C_ALGORITHM(1),
     .C_BYTE_SIZE(9),
     .C_COMMON_CLK(0),
     .C_DEFAULT_DATA("0"),
     .C_DISABLE_WARN_BHV_COLL(1),
     .C_DISABLE_WARN_BHV_RANGE(1),
     .C_FAMILY("virtex5"),
     .C_HAS_ENA(0),
     .C_HAS_ENB(0),
     .C_HAS_MEM_OUTPUT_REGS_A(0),
     .C_HAS_MEM_OUTPUT_REGS_B(0),
     .C_HAS_MUX_OUTPUT_REGS_A(0),
     .C_HAS_MUX_OUTPUT_REGS_B(0),
     .C_HAS_REGCEA(0),
     .C_HAS_REGCEB(0),
     .C_HAS_SSRA(0),
     .C_HAS_SSRB(0),
```

```
      .C_INIT_FILE_NAME("sbox_rom.mif"),
      .C_LOAD_INIT_FILE(1),
      .C_MEM_TYPE(3),
      .C_MUX_PIPELINE_STAGES(0),
      .C_PRIM_TYPE(1),
      .C_READ_DEPTH_A(256),
      .C_READ_DEPTH_B(256),
      .C_READ_WIDTH_A(8),
      .C_READ_WIDTH_B(8),
      .C_SIM_COLLISION_CHECK("NONE"),
      .C_SINITA_VAL("0"),
      .C_SINITB_VAL("0"),
      .C_USE_BYTE_WEA(0),
      .C_USE_BYTE_WEB(0),
      .C_USE_DEFAULT_DATA(0),
      .C_USE_ECC(0),
      .C_USE_RAMB16BWER_RST_BHV(0),
      .C_WEA_WIDTH(1),
      .C_WEB_WIDTH(1),
      .C_WRITE_DEPTH_A(256),
      .C_WRITE_DEPTH_B(256),
      .C_WRITE_MODE_A("WRITE_FIRST"),
      .C_WRITE_MODE_B("WRITE_FIRST"),
      .C_WRITE_WIDTH_A(8),
      .C_WRITE_WIDTH_B(8),
      .C_XDEVICEFAMILY("virtex5"))
    inst (
      .CLKA(clka),
      .ADDRA(addra),
      .DOUTA(douta),
      .DINA(),
      .ENA(),
      .REGCEA(),
      .WEA(),
      .SSRA(),
      .CLKB(),
      .DINB(),
      .ADDRB(),
      .ENB(),
      .REGCEB(),
      .WEB(),
      .SSRB(),
      .DOUTB(),
      .DBITERR(),
      .SBITERR());


// synthesis translate_on

endmodule
```