共通鍵暗号 **AES** 用 **SubBytes** 変換回路の設計

課題　自由課題　**AES** 回路の設計

チーム名　　チーム菅原システムズ

代表者氏名　菅原孝幸（社会人）

設計者　　菅原孝幸(1 名）

所属　　菅原システムズ

住所　　〒981-3215　宮城県仙台市泉区北中山 3-24-13

1.概略

単にコンテスト用回路というだけでなく、FPGA 用 OPENCORE として実用的な回路を設計したいと思いました。そこで、課題の SUBBYTES を含む AES 回路そのものを

・最小の CELL 数
・最速のスループット

で設計することを目標にしました。具体的には、ALTERA FPGA 用の OPENCORE として以下を目標としました。

・CELL 数 2910 以下（2003 年 10 月号付録の CYCLONE と同等以下）
・動作 CLOCK 200MHz スループットとして 2Gbps 以上

AES 回路は、既に OPENCORE(www.opencores.org)から Rudolf Usselmann 氏によるものが公開されております。(下表参照)

Sample Synthesis Results for the Cipher Block

Technology	Size/Area	Speed/Performance
Xilinx Spartan IIe XS2V200-6	3497 LUTs (74 %), 1026 Regs. (21 %)	101 Mhz (1.08 Gbits/sec)
UMC 0.18u Std. Cell	38K Gates	265 Mhz (2.82 Gbits/sec)

Sample Synthesis Results for the Inverse Cipher Block

Technology	Size/Area	Speed/Performance
Xilinx Spartan IIe XS2V200-6	3393 LUTs (72 %), 883 Regs. (18 %)	85 Mhz (906 Mbits/sec)
UMC 0.18u Std. Cell	50K Gates	235 Mhz (2.5 Gbits/sec)

内部的には、
1）Encryption/Decryption 回路は、別個の回路
2）SUBBYTES/逆 SUBBYTES は、各々 ROM で実現
3）パイプラインはなし

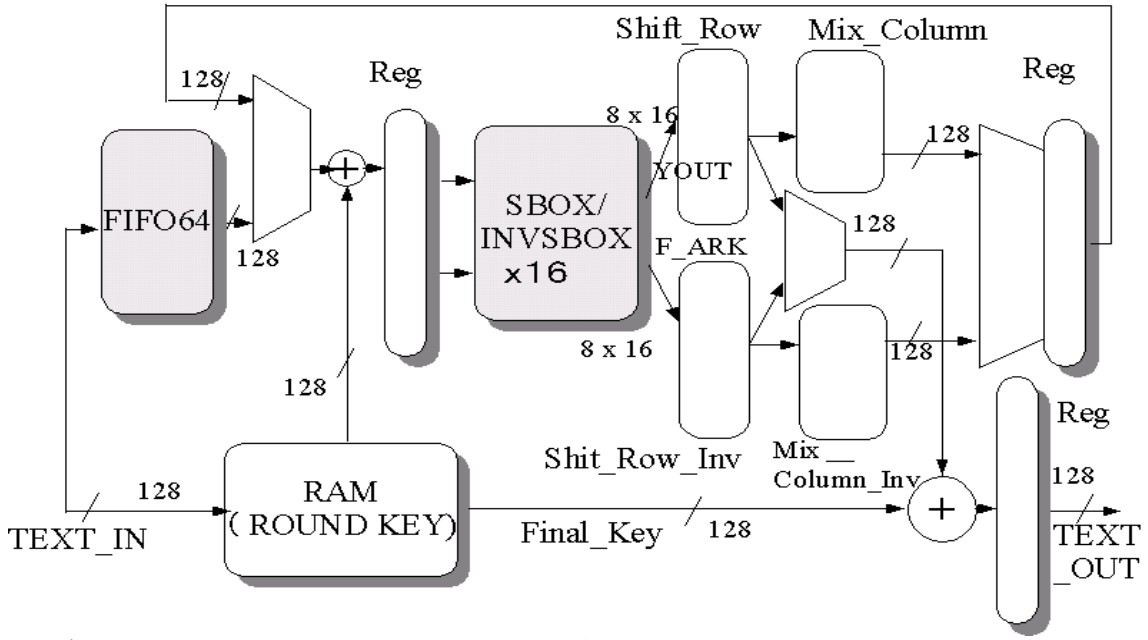
という仕様で実現されています。
そこで、
1）Encryption/Decryption は一つの回路を共用し Cell 数を削減
2）SUBBYTES/逆 SUBBYTES はガロア体の演算手法により少 CELL 化
3）パイプライン手法により動作 CLOCK 向上

という方針で再設計することより目標仕様達成を図りました。

2．回路ブロックの説明

2．1 AES 回路ブロック

AES 回路ブロック図を下に示します。



以下は各 Component および、設計仕様の説明です。

・RAM （128 ビット×32 ワード）

-HOST で生成された ROUND KEY を保持します。本 AES 回路では ROUNDKEY 生成は HOST で行うものとしています。

また、Key Length は 128 ビットのみの仕様とします。

・FIFO64 （128 ビット×64 ワード）

-Host からの PlainText・CipherText の受信 Buffer です。Control 回路はパイプラインを満たすことが可能と判断した場合のみ、パイプラインに Data を流します。逆に、パイプラインを満たせない場合は、FIFO に残ってしまう仕様です。本 AES 回路は任意ワード数のプロセシングが可能ですが、FIFO を最終的に Flush するためには 5 の整数倍のワード総数で終了する必要があります。

・SBOX/INVSBOX -（16 個並列）

-SUBBYTES/逆 SUBBYTES 変換回路を 16 個並列に使用します。

・SHIFT_ROW・SHIFT_ROW_INV

-Shift 動作を行います。ゲートはありません。

・MIX_COLUMN・MIX_COLUMN_INV

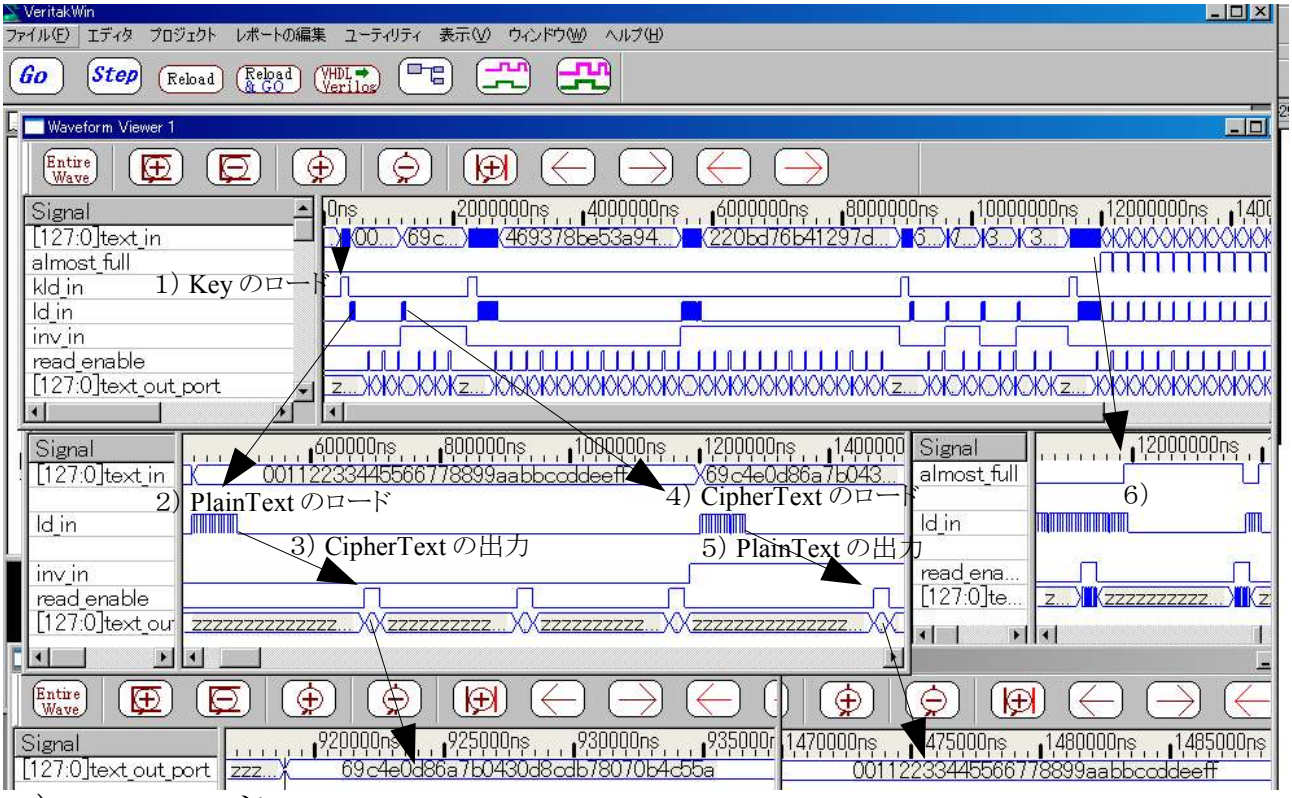
・CONTROL 回路（図示していません）

-パイプライン制御を行います。

SBOX/INVSBOX 内に 3 段の reg があり、AES 回路全体では、5 段のパイプラインとなっています。さらに、最終段に reg を設け、通常11クロックのスループットを 10 クロックに削減しています。

2．2 AES 回路インターフェースタイミング図

機能設計および機能シミュレーションは、自作の Verilog シミュレータを使用しています。シミュレーションの出力波形で説明します。（下図）



1）Key のロード
Host 側で生成した RoundKey（Encryption と Decryption 用の両方）を text_in バスに乗せ kld_in を H にして AES 回路に読みこませます。

2）PlainText のロード
ld_in を H にして text_in バスから AES 回路に読み込ませます。128 ビットを 1ワードとして任意ワード数のロードができます。

3）Encryption の指定
Encryption の指定は、inv_in を L にします。

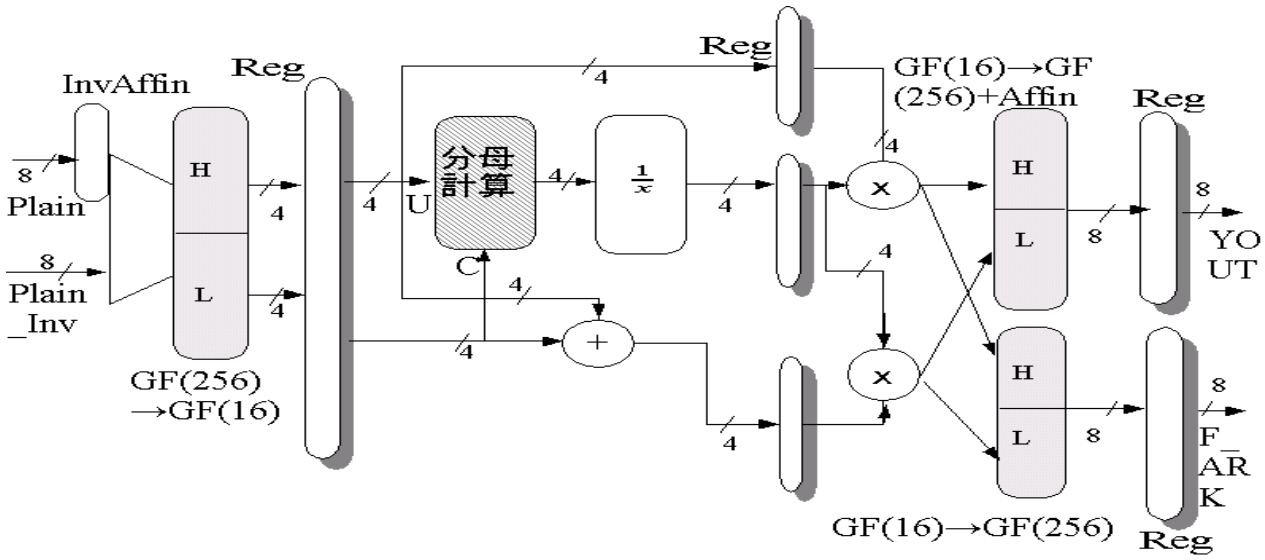
4）CipherText の出力
CipherText の生成が完了すると Read_Enable が H になり text_out_port に結果が出力されます。

5）Decryption
inv_in を H にした後は4）と同様です。Key が同じであるときは、 再度 Key をロードする必要はありません。これは Encryption も同様です。

6）FIFO 制御
almost_full 信号が H のときは、まもなく内蔵 FIFO が Overflow してしまうことを示していますので、Host は一時 load_in の入力を中断し、再び L になるのを待ちます。

上図では、PlainText　0011..が、69c..の CipherText に変換（Encryption）され、69c..が PlainText　0011 に戻る（Decryption）様子を示しています。

2．3　SUBBYTES/逆 SUBBYTES の回路ブロック



（1）変換マトリクス

拡大体を利用し逆元計算を高速化します。すなわち GF(256)の元を GF(16)上の2次元ベクトルとして表現します。

GF(256)上の任意の元は y は　y=a0+ a1 x　(a0,a1 ∈GF(16))
その逆元は　y=b0+b1 x　(b0,b1 ∈GF(16))とすると

b0=(a0+a1)/分母
b1=a1/分母
分母=a0(a0+a1)+β a1 a1

となります。（たとえば、＊文献1によります。）
GF(256)上の多項式　R(x)=　$x^8+x^4+x^3+x+1$　は既約多項式ではありますが、原始多項式ではありません。そこで、GF(256)上の原始根 α を GF(16)上の原始根 γ にマップする方法をとりました。

GF(16)上の既約多項式　　x^4+x+1
GF(16)上の2次原始多項式　　$x^2+x+\beta$
ここで、
β=　γ^{14}　、　α^{238}　=γ
としました。

γ^{14}　を選択したのは、GF(16)上の定数掛け算が簡単になるからです。(後述)
Affin 変換は、GF(16)→GF(256)変換とマージした方が、ゲート数と遅延時間の観点から有利です。homomorphsim を満たす α は、Exhaustive サーチ

によりいくつか見つかりましたが、ゲート数が最小になるものを選択しました。

以上より変換 Matrix は下記のようになります。

GF(256)→GF(16)

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

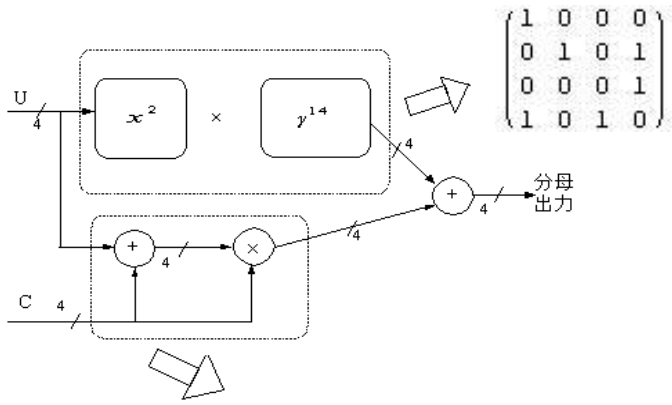
GF(16)→GF(256)

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

GF(16)→GF(256)+Affin

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

(2)分母の計算
自乗項および定数項の乗算部は下のように2個の EXOR で簡単に実現できます。



$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

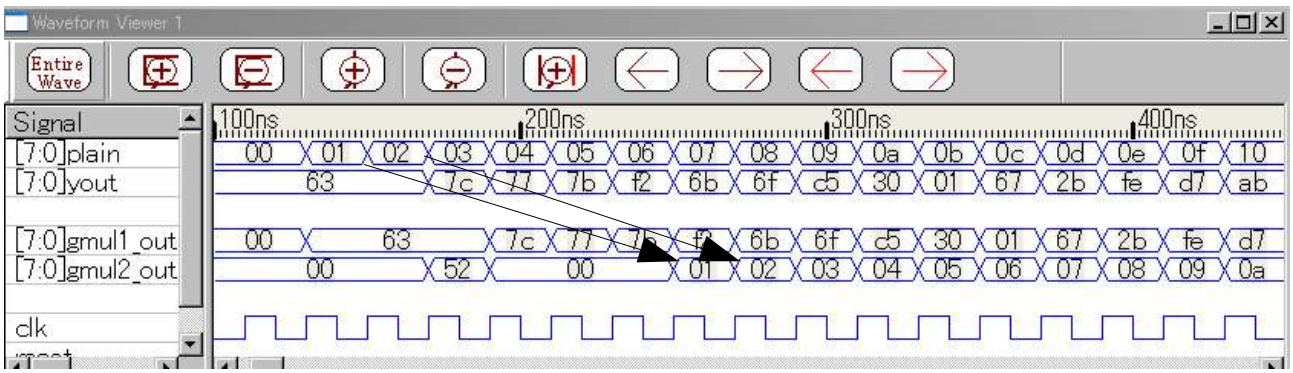
$$c0^2 + c2^2 + c0\,u0 + c3\,u1 + c2\,u2 + c1\,u3 + (c2^2 + c1\,u0 + c0\,u1 + c3\,u1 + c2\,u2 + c3\,u2 + c1\,u3 + c2\,u3)\,x + (c1^2 + c3^2 + c2\,u0 + c1\,u1 + c0\,u2 + c3\,u2 + c2\,u3 + c3\,u3)\,x^2 + (c3^2 + c3\,u0 + c2\,u1 + c1\,u2 + c0\,u3 + c3\,u3)\,x^3$$

変数の乗算部は、クリティカルパスであることを考慮して若干冗長な設計となりますが、上式のように速度優先の記述としました。

(3)入出力
AES 全体回路から考える SUBBYTES/逆 SUBBYTES 各々専用ポートとした方が全体の Selector は削減できゲートディレイの点でも有利です。

(4)機能検証
基本課題1のテストベンチを流用しました。ただし、言語が VHDL なので、自作のトランスレータ VHDL→Verilog によりテストベンチを Verilog に自動変換しました。さらに次の 3 行を加えて完成です。

```
wire [7:0] f_ark1,gmul1_out,gmul2_out,yout2;
    gmul_pipe_ver2 gmul1( clk,plain,8'h00,gmul1_out,inv,reset,f_ark1);
   gmul_pipe_ver2 gmul2( clk,8'h00,gmul1_out,yout2,!inv,reset,gmul2_out);
```



設計した SBOX/INVSBOX を二つ直列につないで PipeLineDelay (3x2=6Clock)後に Plain が元に戻ることが検証できました。

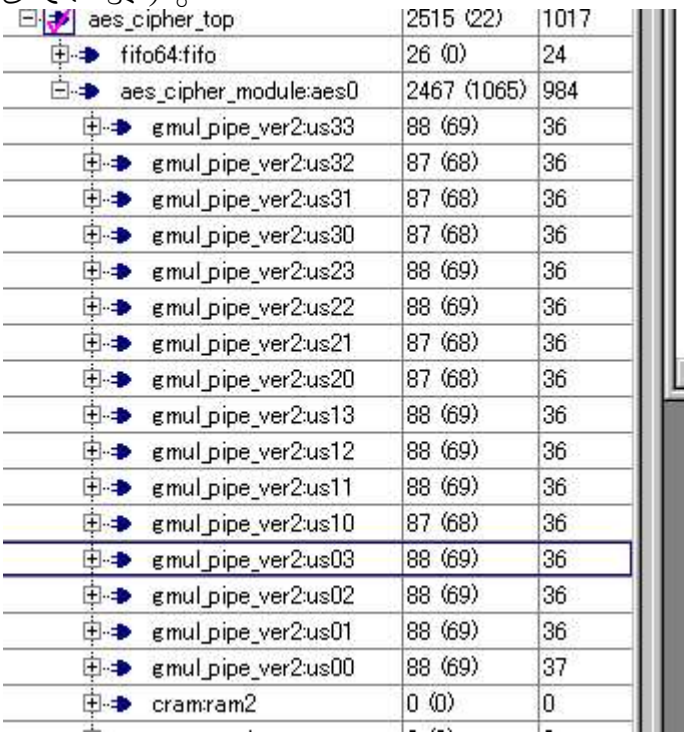
3. 論理合成結果

3. 1　デバイス選択

128 ビットバスが入力・出力に要ります。
FPGA 内バスに接続する用途を想定していますので、外部
ポートに出力する必要はないのですが、Simulation/Debug の容易性から
外部に接続する仕様とします。
そのため、400 ピンの Cyclone　EP1C4F400FC6 を選択しました。

3. 2 Cell 規模

Quartus II Ver.3.0　Sp2 で論理合成しました。
下のように AES 回路全体で 2515Cell となりました。
内、SUBBYTES/逆 SUBBYTES 回路は、16 個ありますが、87-88CELL
で実現しています。



aes_cipher_top	2515 (22)	1017
fifo64:fifo	26 (0)	24
aes_cipher_module:aes0	2467 (1065)	984
gmul_pipe_ver2:us33	88 (69)	36
gmul_pipe_ver2:us32	87 (68)	36
gmul_pipe_ver2:us31	87 (68)	36
gmul_pipe_ver2:us30	87 (68)	36
gmul_pipe_ver2:us23	88 (69)	36
gmul_pipe_ver2:us22	88 (69)	36
gmul_pipe_ver2:us21	87 (68)	36
gmul_pipe_ver2:us20	87 (68)	36
gmul_pipe_ver2:us13	88 (69)	36
gmul_pipe_ver2:us12	88 (69)	36
gmul_pipe_ver2:us11	88 (69)	36
gmul_pipe_ver2:us10	87 (68)	36
gmul_pipe_ver2:us03	88 (69)	36
gmul_pipe_ver2:us02	88 (69)	36
gmul_pipe_ver2:us01	88 (69)	36
gmul_pipe_ver2:us00	88 (69)	37
cram:ram2	0 (0)	0

３．3速度

AES 回路全体では 212MHz となりました。（ただし、入出力は除きます。）

	Type	Slack	Required Time	Actual Time
1	Clock Setup: 'clk'	-0.253 ns	225.02 MHz (period = 4.444 ns)	212.90 MHz (period = 4.697 ns)
2	Worst-case tsu	6.182 ns	9.200 ns	3.018 ns
3	Worst-case tco	N/A	None	8.194 ns
4	Worst-case tpd	N/A	None	4.127 ns
5	Worst-case th	N/A	None	0.046 ns
6	Worst-case minimum tco	N/A	None	5.479 ns
7	Worst-case minimum tpd	N/A	None	4.127 ns

SUBBYTES/逆 SUBBYTES　単独回路では、260MHz（ただし、FF 間）となりました。

	Type	Slack	Required Time	Actual Time
1	Clock Setup: 'clock'	0.599 ns	225.02 MHz (period = 4.444 ns)	260.08 MHz (period = 3.845 ns)
2	Worst-case tsu	6.584 ns	9.200 ns	2.616 ns
3	Worst-case tco	N/A	None	5.513 ns
4	Worst-case th	N/A	None	0.291 ns
5	Worst-case minimum tco	N/A	None	5.234 ns

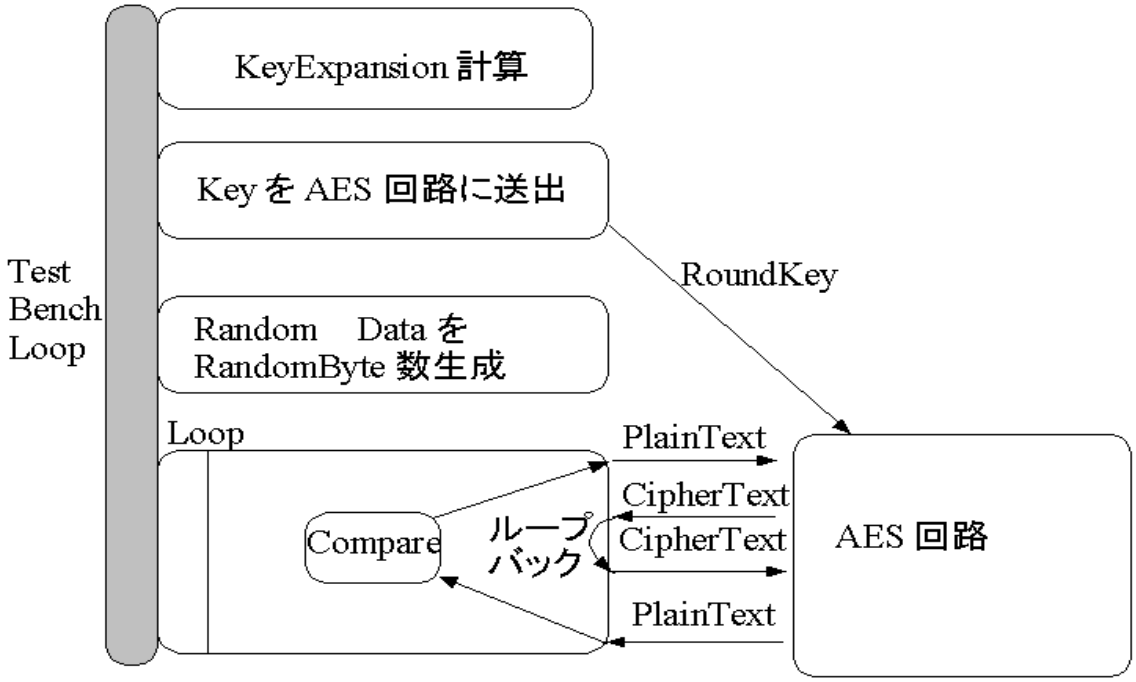
なお、EXOR50 個の回路の合成結果は、Cell 数 17、Speed は、8.06ns (入出力間）となりました。

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time
1	Worst-case tpd	N/A	None	8.060 ns
2	Worst-case minimum tpd	N/A	None	5.500 ns

4．ポストレイアウトシミュレーション

３で論理合成した結果を Verilog に Back Annotation し、添付 8.Verilog Source List に示すテストベンチでテストしました。シミュレーションは MODELSIM で行いました。Clock 周期 4700ps でのテストです。

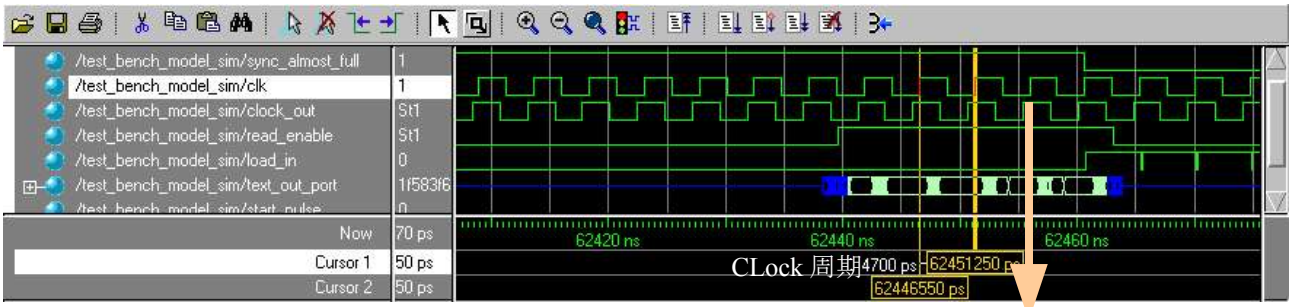


上記テストを行い、Compare Error 発生がないことを確認しました。
以下は、MODELSIM のログリストです。

```
run -all
# Loading aes_cipher_top_v.sdo
# ** Note: (vsim-3587) SDF Backannotation Successfully Completed.
#    Time: 0 ps  Iteration: 0  Instance: /test_bench_model_sim/dut
# ** Error: QuartusIIVersion3.0(E:/modelsim/win32aloem/../altera/Verilog/src/cyclone_atoms.v)(1219): $setup
( data:4735 ps, posedge clk:4773 ps, 72 ps );
#    Time: 4773 ps  Iteration: 1  Instance: /test_bench_model_sim/dut/\aes0|ram1|altsyncram_component|ram_block[0]
[113]\/ram_portadatain_reg
# Testing data_words=        65
# Testing data_words=        10
# Testing data_words=        10
# Fixed length Test Passed.$time=           11283270
# Testing data_words=       370
# Testing data_words=       455
# Testing data_words=       290
# Random #of bytes test Passed.$time=           117005070
# Testing data_words=        65
# Test Done with no errors.          123688470
# ** Note: $finish    : E:/qualtas/tak design/try2/test_bench_model_sim.v(140)
```

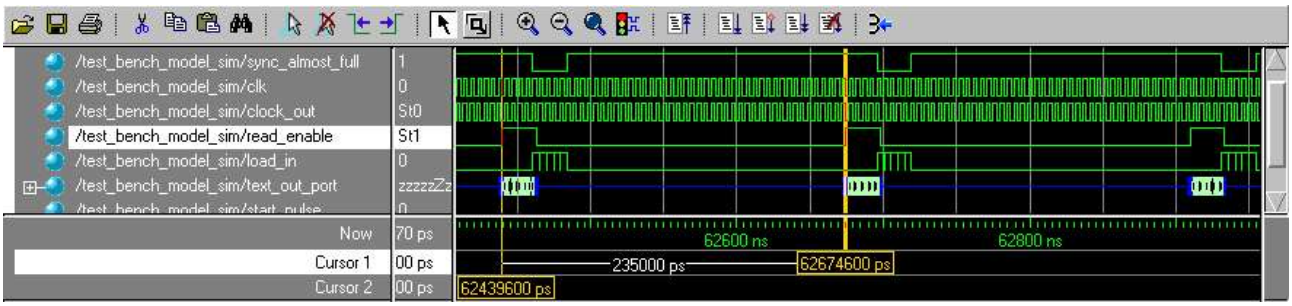
（Initial の Clock エッジで TimingError が発生していますが Reset 中なので問題ありません。）

なお、AES 回路からの出力信号は、IO 遅延の為、1 Clock 以上遅れます。
テストベンチに Data を取り込む Strobe　Timing は波形を見ながら調整しました。



Strobe　Point

5. プロセシングスピード/拡張性
下に Encryption/Decryption を連続で行っているシミュレーションを示します。



Read_Enable 間は、
Encryption/Decryption Throughput :235000/4700=50 Clock
この間に、PipeLine Depth(5)分の処理が出来ますのでスループットは、
1/(50*4700/128/5)=2.72Gbps
となり目標達成です。

拡張性について
本アーキテクチャは、FIFO 部とパイプライン部が module で独立していますので、パイプライン部インスタンスを増やすことで容易にスループットを増強できるでしょう。(x9までは、アーキテクチャ変更なしに
対応可能だと思います。規模的に Web Edition でもx7までは行けます。)
また、パイプライン Flush 等の制御信号を追加したい場合も、TOP 層の module をいじくるだけで済むはずです。

6. まとめ
ALTERA FPGA 用 OPENCORE として AES 回路を設計、機能シミュレーション、論理合成、ポストレイアウトシミュレーションまで行い以下の結果を得ました。

	目標仕様	論理合成後
Cell 数	2910 以下	2515
Throughput	2Gbps 以上	2.72Gbps

7. 感想
Cyclone の Speed には目を見張るものがあります。10 年前にリードソロモンの演算回路を設計したときは、0.5μm ルールの CBIC でしたが、
32MHzClock で苦労した覚えがあり隔世の感があります。
FPGA の世界がこれからどうなってゆくのか本当に楽しみです。

８. Verilog Source List・文献リスト

SBOX/INVSBOX 回路テストベンチ : test_subbytes
AES 回路テストベンチ : test_bench_model_sim
AES 回路 : aes_cipher_top

文献1 : 森井、笠原 GF(2m)における高速算法について
電子通信情報学会技術研究報告 IT87-24

SBOX/INV_SBOX 回路テストベンチ

```
//----------------------------------------------------------------------
//
//   This file is automatically generated by VHDL to Verilog Translator.
//              www.sugawara-systems.com
//                Build Nov.5.2003
//                  tech-support@sugawara-systems.com
//       See Original Copyright Notice for property of the file .( somewhere in this file.)
//
//----------------------------------------------------------------------


// Univ. of the Ryukyus LSI design contest 2004
// SubBytes Transform Circuit for AES Cipher
// file: SubBytes.vhd
// combinational logic of SubBytes transform
// Tom Wada 2003/September/15

`timescale 1ns/1ns
`define ns 1
`define us (1000*`ns)
`define ms (1000*`us)
`define sec (1000*`ms)

module  subbytes ( clk, reset, xin, inv, yout );
    input clk;
    input reset;
    input [7:0]  xin ;
    input inv;
    output [7:0]  yout ;

        reg [7:0] invrom [0:255];
   initial begin
                invrom[0]='b00000000;
                invrom[1]='b00000001;
                invrom[2]='b10001101;
                invrom[3]='b11110110;
                invrom[4]='b11001011;
                invrom[5]='b01010010;
                invrom[6]='b01111011;
                invrom[7]='b11010001;
                invrom[8]='b11101000;
                invrom[9]='b01001111;
                invrom[10]='b00101001;
                invrom[11]='b11000000;
                invrom[12]='b10110000;
                invrom[13]='b11100001;
                invrom[14]='b11100101;
                invrom[15]='b11000111;
                invrom[16]='b01110100;
                invrom[17]='b10110100;
                invrom[18]='b10101010;
                invrom[19]='b01001011;
                invrom[20]='b10011001;
                invrom[21]='b00101011;
                invrom[22]='b01100000;
                invrom[23]='b01011111;
                invrom[24]='b01011000;
                invrom[25]='b00111111;
                invrom[26]='b11111101;
                invrom[27]='b11001100;
                invrom[28]='b11111111;
                invrom[29]='b01000000;
```

```
invrom[30]='b11101110;
invrom[31]='b10110010;
invrom[32]='b00111010;
invrom[33]='b01101110;
invrom[34]='b01011010;
invrom[35]='b11110001;
invrom[36]='b01010101;
invrom[37]='b01001101;
invrom[38]='b10101000;
invrom[39]='b11001001;
invrom[40]='b11000001;
invrom[41]='b00001010;
invrom[42]='b10011000;
invrom[43]='b00010101;
invrom[44]='b00110000;
invrom[45]='b01000100;
invrom[46]='b10100010;
invrom[47]='b11000010;
invrom[48]='b00101100;
invrom[49]='b01000101;
invrom[50]='b10010010;
invrom[51]='b01101100;
invrom[52]='b11110011;
invrom[53]='b00111001;
invrom[54]='b01100110;
invrom[55]='b01000010;
invrom[56]='b11110010;
invrom[57]='b00110101;
invrom[58]='b00100000;
invrom[59]='b01101111;
invrom[60]='b01110111;
invrom[61]='b10111011;
invrom[62]='b01011001;
invrom[63]='b00011001;
invrom[64]='b00011101;
invrom[65]='b11111110;
invrom[66]='b00110111;
invrom[67]='b01100111;
invrom[68]='b00101101;
invrom[69]='b00110001;
invrom[70]='b11110101;
invrom[71]='b01101001;
invrom[72]='b10100111;
invrom[73]='b01100100;
invrom[74]='b10101011;
invrom[75]='b00010011;
invrom[76]='b01010100;
invrom[77]='b00100101;
invrom[78]='b11101001;
invrom[79]='b00001001;
invrom[80]='b11101101;
invrom[81]='b01011100;
invrom[82]='b00000101;
invrom[83]='b11001010;
invrom[84]='b01001100;
invrom[85]='b00100100;
invrom[86]='b10000111;
invrom[87]='b10111111;
invrom[88]='b00011000;
invrom[89]='b00111110;
invrom[90]='b00100010;
invrom[91]='b11110000;
invrom[92]='b01010001;
invrom[93]='b11101100;
```

```
invrom[94]='b01100001;
invrom[95]='b00010111;
invrom[96]='b00010110;
invrom[97]='b01011110;
invrom[98]='b10101111;
invrom[99]='b11010011;
invrom[100]='b01001001;
invrom[101]='b10100110;
invrom[102]='b00110110;
invrom[103]='b01000011;
invrom[104]='b11110100;
invrom[105]='b01000111;
invrom[106]='b10010001;
invrom[107]='b11011111;
invrom[108]='b00110011;
invrom[109]='b10010011;
invrom[110]='b00100001;
invrom[111]='b00111011;
invrom[112]='b01111001;
invrom[113]='b10110111;
invrom[114]='b10010111;
invrom[115]='b10000101;
invrom[116]='b00010000;
invrom[117]='b10110101;
invrom[118]='b10111010;
invrom[119]='b00111100;
invrom[120]='b10110110;
invrom[121]='b01110000;
invrom[122]='b11010000;
invrom[123]='b00000110;
invrom[124]='b10100001;
invrom[125]='b11111010;
invrom[126]='b10000001;
invrom[127]='b10000010;
invrom[128]='b10000011;
invrom[129]='b01111110;
invrom[130]='b01111111;
invrom[131]='b10000000;
invrom[132]='b10010110;
invrom[133]='b01110011;
invrom[134]='b10111110;
invrom[135]='b01010110;
invrom[136]='b10011011;
invrom[137]='b10011110;
invrom[138]='b10010101;
invrom[139]='b11011001;
invrom[140]='b11110111;
invrom[141]='b00000010;
invrom[142]='b10111001;
invrom[143]='b10100100;
invrom[144]='b11011110;
invrom[145]='b01101010;
invrom[146]='b00110010;
invrom[147]='b01101101;
invrom[148]='b11011000;
invrom[149]='b10001010;
invrom[150]='b10000100;
invrom[151]='b01110010;
invrom[152]='b00101010;
invrom[153]='b00010100;
invrom[154]='b10011111;
invrom[155]='b10001000;
invrom[156]='b11111001;
invrom[157]='b11011100;
```

```
invrom[158]='b10001001;
invrom[159]='b10011010;
invrom[160]='b11111011;
invrom[161]='b01111100;
invrom[162]='b00101110;
invrom[163]='b11000011;
invrom[164]='b10001111;
invrom[165]='b10111000;
invrom[166]='b01100101;
invrom[167]='b01001000;
invrom[168]='b00100110;
invrom[169]='b11001000;
invrom[170]='b00010010;
invrom[171]='b01001010;
invrom[172]='b11001110;
invrom[173]='b11100111;
invrom[174]='b11010010;
invrom[175]='b01100010;
invrom[176]='b00001100;
invrom[177]='b11100000;
invrom[178]='b00011111;
invrom[179]='b11101111;
invrom[180]='b00010001;
invrom[181]='b01110101;
invrom[182]='b01111000;
invrom[183]='b01110001;
invrom[184]='b10100101;
invrom[185]='b10001110;
invrom[186]='b01110110;
invrom[187]='b00111101;
invrom[188]='b10111101;
invrom[189]='b10111100;
invrom[190]='b10000110;
invrom[191]='b01010111;
invrom[192]='b00001011;
invrom[193]='b00101000;
invrom[194]='b00101111;
invrom[195]='b10100011;
invrom[196]='b11011010;
invrom[197]='b11010100;
invrom[198]='b11100100;
invrom[199]='b00001111;
invrom[200]='b10101001;
invrom[201]='b00100111;
invrom[202]='b01010011;
invrom[203]='b00000100;
invrom[204]='b00011011;
invrom[205]='b11111100;
invrom[206]='b10101100;
invrom[207]='b11100110;
invrom[208]='b01111010;
invrom[209]='b00000111;
invrom[210]='b10101110;
invrom[211]='b01100011;
invrom[212]='b11000101;
invrom[213]='b11011011;
invrom[214]='b11100010;
invrom[215]='b11101010;
invrom[216]='b10010100;
invrom[217]='b10001011;
invrom[218]='b11000100;
invrom[219]='b11010101;
invrom[220]='b10011101;
invrom[221]='b11111000;
```

```
            invrom[222]='b10010000;
            invrom[223]='b01101011;
            invrom[224]='b10110001;
            invrom[225]='b00001101;
            invrom[226]='b11010110;
            invrom[227]='b11101011;
            invrom[228]='b11000110;
            invrom[229]='b00001110;
            invrom[230]='b11001111;
            invrom[231]='b10101101;
            invrom[232]='b00001000;
            invrom[233]='b01001110;
            invrom[234]='b11010111;
            invrom[235]='b11100011;
            invrom[236]='b01011101;
            invrom[237]='b01010000;
            invrom[238]='b00011110;
            invrom[239]='b10110011;
            invrom[240]='b01011011;
            invrom[241]='b00100011;
            invrom[242]='b00111000;
            invrom[243]='b00110100;
            invrom[244]='b01101000;
            invrom[245]='b01000110;
            invrom[246]='b00000011;
            invrom[247]='b10001100;
            invrom[248]='b11011101;
            invrom[249]='b10011100;
            invrom[250]='b01111101;
            invrom[251]='b10100000;
            invrom[252]='b11001101;
            invrom[253]='b00011010;
            invrom[254]='b01000001;
            invrom[255]='b00011100;

end//initial

    reg [7:0] invout;
 reg [7:0] affine;
 wire [7:0] yout;


always @ (posedge clk ) begin
            begin
                if ((reset === 1'b1))
                    invout <= 'b00000000;

                else
                    invout <= invrom[xin];



            end

end //always

always @ (posedge clk ) begin
            begin
                affine[7] <= (((((invout[7] ^ invout[6]) ^ invout[5]) ^ invout[4]) ^ invout[3]) ^ 1'b0);
                affine[6] <= (((((invout[6] ^ invout[5]) ^ invout[4]) ^ invout[3]) ^ invout[2]) ^ 1'b1);
                affine[5] <= (((((invout[5] ^ invout[4]) ^ invout[3]) ^ invout[2]) ^ invout[1]) ^ 1'b1);
                affine[4] <= (((((invout[4] ^ invout[3]) ^ invout[2]) ^ invout[1]) ^ invout[0]) ^ 1'b0);
                affine[3] <= (((((invout[7] ^ invout[3]) ^ invout[2]) ^ invout[1]) ^ invout[0]) ^ 1'b0);
                affine[2] <= (((((invout[7] ^ invout[6]) ^ invout[2]) ^ invout[1]) ^ invout[0]) ^ 1'b0);
                affine[1] <= (((((invout[7] ^ invout[6]) ^ invout[5]) ^ invout[1]) ^ invout[0]) ^ 1'b1);
```

```
                affine[0] <= (((((invout[7] ^ invout[6]) ^ invout[5]) ^ invout[4]) ^ invout[0]) ^ 1'b1);
            end


   end //always
   assign {yout}=affine;


endmodule
// Univ. of the Ryukyus LSI design contest 2004
// SubBytes Transform Circuit for AES Cipher
// file: sender.vhd
// Sender generates 8 bit integer from 0 to 255
// Tom Wada 2003/September/15


module  sender ( clk, reset, plain );
    input clk;
    input reset;
    output [7:0]  plain ;

    reg [7:0] count;
    wire [7:0] plain;


   always @ (posedge clk ) begin
            begin
                if ((reset === 1'b1))
                    count <= 'b00000000;

                else
                    count <= (count + 1);



            end

   end //always
   assign {plain}=count;


endmodule
// Univ. of the Ryukyus LSI design contest 2004
// SubBytes Transform Circuit for AES Cipher
// file: test_subbytes.vhd
// TESTBENCH
// Tom Wada 2003/September/15


module  test_subbytes; // Null Port List
    reg clk;
   initial clk = 1'b0;
    reg reset;
   initial reset = 1'b1;
    integer cycle = 0;
    wire [7:0] plain;
    wire [7:0] yout;
    reg inv;
   initial inv = 1'b0;
   always begin
          if ((cycle < 1000))
            begin
                cycle <= (cycle + 1);
                #(10* `ns);
                clk <= ~ (clk);
```

```
            end
        else
            wait(0);


    end
    always begin
        begin
            begin :Block_Name_1
                integer n;
                for (n=0;n<=5;n=n+1) begin
                    @(negedge clk );

                end //for
            end //end Block
            reset <= 1'b0;
        end
    end
    sender i_sender(.clk(clk), .reset(reset), .plain(plain));
    subbytes i_subbytes(.clk(clk), .reset(reset), .xin(plain), .inv(inv), .yout(yout));
//Addition begin
// 設計した SBOX/INVSBOX を二つ直列につないで PipeLineDelay（3x2=6 Clock）後に
// 元に戻ることを検証する。
    wire [7:0] f_ark1,gmul1_out,gmul2_out,yout2;

    gmul_pipe_ver2 gmul1( clk,plain,8'h00,gmul1_out,inv,reset,f_ark1);
    gmul_pipe_ver2 gmul2( clk,8'h00,gmul1_out,yout2,!inv,reset,gmul2_out);


 //Addition end
endmodule
```

AES 回路テストベンチ

```
`timescale 1ps/1ps


///   AES CIPHER CORE for Altera FPGA CYCLONE
///   This is derivative work from original code of Rudolf Usselmann.
///   Almost all codes and Interface are changed from the original.
///
///   Copyright (C) 2003 Tak.Sugawara
///    Nov.7.2003
///    Author: Tak.Sugawara
///    www.sugawara-systems.com
///    tech-support@sugawara-systems.com
///
///
///
///
///
//////////////////////////////////////////////////////////////////////
////                                                              ////
////  AES Test Bench                                              ////
////                                                              ////
////                                                              ////
////  Author: Rudolf Usselmann                                    ////
////          rudi@asics.ws                                       ////
////                                                              ////
////                                                              ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/  ////
////                                                              ////
//////////////////////////////////////////////////////////////////////
////                                                              ////
//// Copyright (C) 2000-2002 Rudolf Usselmann                     ////
////                         www.asics.ws                         ////
////                         rudi@asics.ws                        ////
////                                                              ////
//// This source file may be used and distributed without         ////
//// restriction provided that this copyright statement is not    ////
//// removed from the file and that any derivative work contains  ////
//// the original copyright notice and the associated disclaimer.////
////                                                              ////
////     THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY      ////
//// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED    ////
//// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS    ////
//// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR       ////
//// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,          ////
//// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES     ////
//// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE    ////
//// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR         ////
//// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF   ////
//// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT   ////
//// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT   ////
//// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE          ////
//// POSSIBILITY OF SUCH DAMAGE.                                  ////
////                                                              ////
//////////////////////////////////////////////////////////////////////

//  CVS Log
//
//  $Id: test_bench_top.v,v 1.2 2002/11/12 16:10:12 rudi Exp $
//
//  $Date: 2002/11/12 16:10:12 $
//  $Revision: 1.2 $
```

```
//  $Author: rudi $
//  $Locker:  $
//  $State: Exp $
//
// Change History:
//              $Log: test_bench_top.v,v $
//              Revision 1.2  2002/11/12 16:10:12  rudi
//
//              Improved test bench, added missing timescale file.
//
//              Revision 1.1.1.1  2002/11/09 11:22:56  rudi
//              Initial Checkin
//
//
//
//
//
//

module test_bench_model_sim;
            reg clk=0;
            reg reset=0;
            reg load_in;
            reg start_pulse;

            wire done;
            reg [127:0] text_in=0;
            reg kld_in=0;
            reg inv_in=0;
            integer i,k;
            reg [127:0] plain_text[0:10000] ;//=128'h00112233445566778899aabbccddeeff;
            reg [127:0] key=128'h000102030405060708090a0b0c0d0e0f;
            reg [127:0] ciphertext[0:10000] ;// 128'h69c4e0d86a7b0430d8cdb78070b4c55a;
            reg [127:0] received_data[0:10000];

            reg [127:0] key_buffer[0:31];
            reg [127:0] key_inv_buffer[0:31];
            wire [127:0] text_out_port;
            reg kld;

            wire [31:0] w0,w1,w2,w3;
            wire [31:0] wr0,wr1,wr2,wr3;
            wire key_load_done;
            wire [5:0] usedw;
            wire clock_out,almost_full;

wire        [7:0]     sa00_ark, sa01_ark, sa02_ark, sa03_ark;
wire        [7:0]     sa10_ark, sa11_ark, sa12_ark, sa13_ark;
wire        [7:0]     sa20_ark, sa21_ark, sa22_ark, sa23_ark;
wire        [7:0]     sa30_ark, sa31_ark, sa32_ark, sa33_ark;

reg         [3:0]     kcnt;
reg kb_ld;
reg                   kdone;
reg   [31:0] received_word_length,send_data_length;
integer errors=0;
wire      data_valid;
wire fifo_read_enable;
wire [4:0] kcounter;
wire fifo_write_enable;
//Test Bench Starts
            initial begin
                        reset_env;
                        simple_test1;
```

```
                random_test1(1);

                random_test3(2);
                if (!errors) $display("Fixed length Test Passed.$time=%d",$time);

                random_test2(3);
                if (!errors) $display("Random #of bytes test Passed.$time=%d",$time);

                random_test1(1);
                if (!errors) $display("Test Done with no errors.%d",$time);
                #(1*1000) $finish;
        end
//Test Bench Finished.
//`define VERITAK
`define DELAY_SIM
`define HOLD_TIME 100
`define CYCLE_TIME 2350 //
`ifdef VERITAK
        always  #(10) clk<=~clk;
`else
        always  #(`CYCLE_TIME) clk<=~clk;//
`endif
        aes_cipher_top  dut(.clk(clk), .rst(reset), .ld_in(load_in), .done(done), .text_in(text_in), .text_out_port
(text_out_port),.inv_in(inv_in),.kld_in(kld_in),.start_pulse_in(start_pulse),.read_enable(read_enable),.kdone
(key_load_done),.almost_full(almost_full),.clock_out(clock_out));


        task reset_env;
                begin
                        start_pulse=0;
                        kld_in=0;
                        reset=0;
                        load_in=0;
                        text_in=0;
                        inv_in=0;

                        #(1*1005) reset=1;
                        #(1*1000);
                end
        endtask
        task data_length_check;
                input [31:0] bytes;
                begin
                    if (bytes%5 !==0 ) begin
                                $display("Send-bytes modulo 5 should be 0 in order to fill out pipe-line.");
                                $finish;
                        end


                end
        endtask

        task make_random_text_data;
                input  [31:0] bytes;
                integer m3,m2,m1,m0;
                integer n;
                begin
                        data_length_check(bytes);
                        for (n=0; n<bytes; n=n+1) begin
                                m0=$random;
                                m1=$random;
                                m2=$random;
                                m3=$random;
                                plain_text[n]={m3,m2,m1,m1};
```

```
		end
	end
endtask

task simple_test1;
	integer i;
	begin
		key=128'h000102030405060708090a0b0c0d0e0f;
		make_round_keys;
		send_round_keys;
		send_data_length=15;
		for (i=0; i< 64;i=i+1)  plain_text[i] =128'h00112233445566778899aabbccddeeff;

		plain_text[0] =128'h00112233445566778899aabbccddeeff;
		plain_text[1] =128'h00112233445566778899aabbccddeeff;
		plain_text[2] =128'h00112233445566778899aabbccddeeff;
		plain_text[3] =128'h00112233445566778899aabbccddeeff;
		plain_text[4] =128'h00112233445566778899aabbccddeeff;
		plain_text[5] =128'h00112233445566778899aabbccddeeff;
		plain_text[6] =128'h00112233445566778899aabbccddeeff;
		plain_text[7] =128'h00112233445566778899aabbccddeeff;
		plain_text[8] =128'h00112233445566778899aabbccddeeff;
		plain_text[9] =128'h00112233445566778899aabbccddeeff;



		for (i=0; i< 64;i=i+1)	ciphertext[i] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;
		ciphertext[0] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;
		ciphertext[1] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;
		ciphertext[2] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;
		ciphertext[3] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;
		ciphertext[4] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;
		ciphertext[5] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;
		ciphertext[6] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;
		ciphertext[7] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;
		ciphertext[8] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;
		ciphertext[9] =128'h69c4e0d86a7b0430d8cdb78070b4c55a;

		send_plain_texts(send_data_length);
		send_ciphered_text(send_data_length);
		compare(send_data_length);

	end
endtask

task random_test1;
	input [31:0] loop_len;
	integer k;
	integer data_words;
	begin
		key=128'h000102030405060708090a0b0c0d0e0f;
		make_round_keys;
		send_round_keys;
		for (k=0;k< loop_len;k=k+1) begin
			data_words=65;//get_random_num_words(64);
			$display("Testing data_words=%d",data_words);
			make_random_text_data(data_words);
			send_plain_texts(data_words);
			send_back_received_data(data_words);
			compare(data_words);
		end
	end
endtask
```

```
	task random_test2;
		input [31:0] loop_len;
		integer k;
		integer data_words;
		begin
//			key=128'h000102030405060708090a0b0c0d0e0f;
			key=128'habcdefffeeddccbbaa99887766554433;
			make_round_keys;
			send_round_keys;
			for (k=0;k< loop_len;k=k+1) begin
				data_words=get_random_num_words(500);
				$display("Testing data_words=%d",data_words);
				make_random_text_data(data_words);
				send_plain_texts(data_words);
				send_back_received_data(data_words);
				compare(data_words);
			end
		end
	endtask
	task random_test3;//10bytes 固定
		input [31:0] loop_len;
		integer k;
		integer data_words;
		begin
	//		key=128'h01231200010203040506070809a0b0c0d0e0f;
			key=128'h990123495867df7ee727375970bcdeafef4326;

			make_round_keys;
			send_round_keys;
			for (k=0;k< loop_len;k=k+1) begin
				data_words=10;
				$display("Testing data_words=%d",data_words);
				make_random_text_data(data_words);
				send_plain_texts(data_words);
				send_back_received_data(data_words);
				compare(data_words);
			end
		end
	endtask

	function [31:0] get_random_num_words;//
		input [31:0] max_words;
		integer i;
		begin
			i=$random;
			while(i %5 !=0 || max_words<i || i <0) begin//Cyclone FIFO : do not write when remaining
words are less than two.
				i=($random)%(max_words+1);
			end
			get_random_num_words=i;

		end
	endfunction



	task compare;
		input  [31:0] compare_bytes;
		integer k;
		begin
			for (k=0; k<compare_bytes;k=k+1) begin
				if (received_data[k] !==plain_text[k]) begin
					$display("Data Compare Error.! [%d]",k);
					$display("Decoded  Data=%h",received_data[k]);
```

```
                                        $display("Original Data=%h",plain_text[k]);
                                        $display(" ");
                                        $finish;
                                        errors =errors+1;
                                end
                        end
                end
endtask

task set_encode_mode;
        begin

                @(negedge clk);
                inv_in=0;
                @(negedge clk);
                start_pulse=1;
                @(negedge clk);
                start_pulse=0;
                @(negedge clk);

        end
endtask
task set_decode_mode;
        begin
                @(negedge clk);
                inv_in=1;
                @(negedge clk);
                start_pulse=1;
                @(negedge clk);
                start_pulse=0;
                @(negedge clk);

        end
endtask
reg sync_almost_full;
always @(posedge clk) sync_almost_full<=almost_full;

task send_plain_texts;
        input  [31:0] send_bytes;
        integer k;
        begin
                //text の送出
                data_length_check(send_bytes);
                set_encode_mode;
                //send_round_keys_for_encode;
                received_word_length=0;//reset receive buffer pointer
                for (k=0; k<send_bytes; k=k+1)begin

                        @(posedge clk);
                        `ifdef DELAY_SIM #(`HOLD_TIME);//
                          load_in<=0;
                        `endif
                        wait(!sync_almost_full);
                        `ifdef DELAY_SIM #(`HOLD_TIME);//
                          load_in<=1;
                        `endif


                        text_in<=plain_text[k];
                end
                @(posedge clk);
                load_in<=0;
                @(negedge clk);
```

```
                        @(posedge done);
                        //encode done
                        if (received_word_length !=send_bytes) begin
                                $display("Sorry,Length Error. Please debug by yourself. %d %
d",received_word_length,send_bytes);
                        end
                end
        endtask

        task send_back_received_data;
                input  [31:0] send_bytes;
                integer k;
                begin
                        //text の送出
                        data_length_check(send_bytes);
                        set_decode_mode;
                        //send_round_keys_for_decode;
                        received_word_length=0;//reset receive buffer pointer

                        for (k=0; k<send_bytes; k=k+1)begin

                                @(posedge clk);
                                `ifdef DELAY_SIM #(`HOLD_TIME);//
                                  load_in<=0;
                                `endif
                                wait(!sync_almost_full);
                                `ifdef DELAY_SIM #(`HOLD_TIME);//
                                  load_in<=1;
                                `endif


                                text_in<=received_data[k];
                        end
                        @(posedge clk);
                        load_in<=0;
                        @(negedge clk);

                        @(posedge done);
                        //decode done
                        if (received_word_length !=send_bytes) begin
                                $display("Sorry,Length Error. Please debug by yourself. %d %
d",received_word_length,send_bytes);

                        end
                end
        endtask
        task send_ciphered_text;
                input  [31:0] send_bytes;
                integer k;
                begin
                        //text の送出
                        data_length_check(send_bytes);
                        set_decode_mode;
                        //send_round_keys_for_decode;
                        received_word_length=0;//reset receive buffer pointer

                        for (k=0; k<send_bytes; k=k+1)begin

                                @(posedge clk);
                                `ifdef DELAY_SIM #(`HOLD_TIME);//Add 50ps for hold time
                                  load_in<=0;
                                `endif
                                wait(!sync_almost_full);
                                `ifdef DELAY_SIM #(`HOLD_TIME);//Add 50ps for hold time
```

```
                        load_in<=1;
                    `endif
                    text_in<=ciphertext[k];
                end
                @(posedge clk);
                load_in<=0;
                @(negedge clk);

                @(posedge done);
                //decode done
                if (received_word_length !=send_bytes) begin
                    $display("Sorry,Length Error. Please debug by yourself. %d %
d",received_word_length,send_bytes);

                end
            end
        endtask
        task make_round_keys;
            begin
                key_buffer[0]<=0;//
                key_inv_buffer[0]<=0;//

                @(negedge clk);
                //ラウンドキーの生成11個のラウンドキーをRAMにKey_Bufferに収納
                //同時にINVERSEも計算する。ROUND1/11以外は、KEYに対するINV_MIX_COLも計
算する。収納順もINVERSEする。
                kld=1;
                @(negedge clk);
                kld=0;
                `ifdef DELAY_SIM
                  #(1000*(20*12));
                `else
                  #(1*(20*12));
                `endif
                //生成待ち


            end
        endtask

        task send_round_keys;
            begin

            //      Key送出時は、Encodeにすること。
                set_encode_mode;
                //Encode keyの送出
                for (k=1; k<12; k=k+1)begin
                    @(posedge clk);
                `ifdef DELAY_SIM #`HOLD_TIME;//Add 50ps for hold time
                    `endif
                    kld_in<=1;
                    text_in<=key_buffer[k];
                end
                for (k=0; k<4; k=k+1)begin//dummy
                    @(posedge clk);
                    `ifdef DELAY_SIM #`HOLD_TIME;
                    `endif //for hold time
                    kld_in<=1;
                    text_in<=0;
                end


            //Decode keyの送出
                for (k=0; k<12; k=k+1)begin
```

```
                                @(posedge clk);
                                `ifdef DELAY_SIM #`HOLD_TIME;
                                `endif //for hold time
                                kld_in<=1;
                                text_in<=key_inv_buffer[k];

                        end

                        @(posedge clk);
                        `ifdef DELAY_SIM #`HOLD_TIME;
                        `endif //for hold time

                        kld_in<=0;
                        wait( key_load_done);

                end
        endtask




always @(posedge clk) begin
        `ifdef DELAY_SIM
                        #(100);//
                `endif
        if (read_enable) begin

                received_data[received_word_length]<=text_out_port;
                received_word_length=received_word_length+1;
        end
end


always @(posedge clk)
        if(!reset)          kcnt <=  4'h1;
        else
        if(kld)             kcnt <=  4'h1;
        else
        if(kb_ld)           kcnt <=  kcnt + 4'h1;

 always @(posedge clk)
        if(!reset)          kb_ld <=  1'b0;
        else
        if(kld)         kb_ld <=  1'b1;
        else
        if(kcnt==4'hb)  kb_ld <=  1'b0;

 always @(posedge clk)  kdone <=  (kcnt==4'hb) & !kld;


 aes_key_expand_128 u0(
        .clk(           clk     ),
        .kld(           kld),
        .key(           key     ),
        .wo_0(          w0      ),
        .wo_1(          w1      ),
        .wo_2(          w2      ),
        .wo_3(          w3      ));

assign sa00_ark= w0[31:24];
assign sa10_ark= w0[23:16];
assign sa20_ark= w0[15:08];
assign sa30_ark= w0[07:00];
```

```
assign sa01_ark= w1[31:24];
assign sa11_ark= w1[23:16];
assign sa21_ark= w1[15:08];
assign sa31_ark= w1[07:00];


assign sa02_ark= w2[31:24];
assign sa12_ark= w2[23:16];
assign sa22_ark= w2[15:08];
assign sa32_ark= w2[07:00];

assign sa03_ark= w3[31:24];
assign sa13_ark= w3[23:16];
assign sa23_ark= w3[15:08];
assign sa33_ark= w3[07:00];









assign wr0 = inv_mix_col(sa00_ark,sa10_ark,sa20_ark,sa30_ark);
assign wr1 = inv_mix_col(sa01_ark,sa11_ark,sa21_ark,sa31_ark);
assign wr2 = inv_mix_col(sa02_ark,sa12_ark,sa22_ark,sa32_ark);
assign wr3 = inv_mix_col(sa03_ark,sa13_ark,sa23_ark,sa33_ark);

function [31:0] inv_mix_col;
input     [7:0]     s0,s1,s2,s3;
begin
inv_mix_col[31:24]=pmul_e(s0)^pmul_b(s1)^pmul_d(s2)^pmul_9(s3);
inv_mix_col[23:16]=pmul_9(s0)^pmul_e(s1)^pmul_b(s2)^pmul_d(s3);
inv_mix_col[15:08]=pmul_d(s0)^pmul_9(s1)^pmul_e(s2)^pmul_b(s3);
inv_mix_col[07:00]=pmul_b(s0)^pmul_d(s1)^pmul_9(s2)^pmul_e(s3);
end
endfunction

// Some synthesis tools don't like xtime being called recursevly ...
function [7:0] pmul_e;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_e=eight^four^two;
end
endfunction

function [7:0] pmul_9;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_9=eight^b;
end
endfunction

function [7:0] pmul_d;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_d=eight^four^b;
end
endfunction
```

```
function [7:0] pmul_b;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_b=eight^two^b;
end
endfunction

function [7:0] xtime;
input [7:0] b;xtime={b[6:0],1'b0}^(8'h1b&{8{b[7]}});
endfunction

 always @(posedge clk)    if(kb_ld) begin
          key_buffer[kcnt] <= {w0,w1,w2,w3};
          if (kcnt==11 || kcnt==1 )
                    key_inv_buffer[12-kcnt]={w0,w1,w2,w3};
          else      key_inv_buffer[12-kcnt]={wr0,wr1,wr2,wr3};

 end

endmodule

module aes_key_expand_128(clk, kld, key, wo_0, wo_1, wo_2, wo_3);
input             clk;
input             kld;
input     [127:0] key;
output    [31:0]  wo_0, wo_1, wo_2, wo_3;
reg       [31:0]  w[3:0];
wire      [31:0]  tmp_w;
wire      [31:0]  subword;
wire      [31:0]  rcon;

assign wo_0 = w[0];
assign wo_1 = w[1];
assign wo_2 = w[2];
assign wo_3 = w[3];
always @(posedge clk)     w[0] <=  kld ? key[127:096] : w[0]^subword^rcon;
always @(posedge clk)     w[1] <=  kld ? key[095:064] : w[0]^w[1]^subword^rcon;
always @(posedge clk)     w[2] <=  kld ? key[063:032] : w[0]^w[2]^w[1]^subword^rcon;
always @(posedge clk)     w[3] <=  kld ? key[031:000] : w[0]^w[3]^w[2]^w[1]^subword^rcon;
assign tmp_w = w[3];


non_pipelined_gmul u0(    .plain(tmp_w[23:16]), .yout(subword[31:24]),.inv(0));
non_pipelined_gmul u1(    .plain(tmp_w[15:08]), .yout(subword[23:16]),.inv(0));
non_pipelined_gmul u2(    .plain(tmp_w[07:00]), .yout(subword[15:08]),.inv(0));
non_pipelined_gmul u3(    .plain(tmp_w[31:24]), .yout(subword[07:00]),.inv(0));


aes_rcon r0(        .clk(clk), .kld(kld), .out(rcon));
endmodule

//////////////////////////////////////////////////////////////////
////                                                          ////
////  AES RCON Block                                          ////
////                                                          ////
////                                                          ////
////  Author: Rudolf Usselmann                                ////
////          rudi@asics.ws                                   ////
////                                                          ////
////                                                          ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/  ////
////                                                          ////
//////////////////////////////////////////////////////////////////
```

// CVS Log
//
// $Id: aes_rcon.v,v 1.1.1.1 2002/11/09 11:22:38 rudi Exp $
//
// $Date: 2002/11/09 11:22:38 $
// $Revision: 1.1.1.1 $
// $Author: rudi $
// $Locker: $
// $State: Exp $
//
// Change History:
// $Log: aes_rcon.v,v $
//   Revision 1.1.1.1 2002/11/09 11:22:38 rudi
//   Initial Checkin
//
//
//
//
//
//


```
module aes_rcon(clk, kld, out);
input           clk;
input           kld;
output  [31:0]  out;
reg     [31:0]  out;
reg     [3:0]   rcnt;
wire    [3:0]   rcnt_next;

always @(posedge clk)
        if(kld)         out <=  32'h01_00_00_00;
        else            out <=  frcon(rcnt_next);

assign rcnt_next = rcnt + 4'h1;
always @(posedge clk)
        if(kld)         rcnt <=  4'h0;
        else            rcnt <=  rcnt_next;
```

```
function [31:0]    frcon;
input     [3:0]     i;
case(i)   // synopsys parallel_case
   4'h0: frcon=32'h01_00_00_00;
   4'h1: frcon=32'h02_00_00_00;
   4'h2: frcon=32'h04_00_00_00;
   4'h3: frcon=32'h08_00_00_00;
   4'h4: frcon=32'h10_00_00_00;
   4'h5: frcon=32'h20_00_00_00;
   4'h6: frcon=32'h40_00_00_00;
   4'h7: frcon=32'h80_00_00_00;
   4'h8: frcon=32'h1b_00_00_00;
   4'h9: frcon=32'h36_00_00_00;
   default: frcon=32'h00_00_00_00;
endcase
endfunction

endmodule
module non_pipelined_gmul(plain,yout,inv);
          input [7:0] plain;
          input inv;
          output [7:0] yout;
        wire [7:0] uv,affin_out;

          wire [7:0] u16,u16c, cu16;
          wire [3:0] pow2,adder1,adder2,x14;
          wire [3:0] m0_A,m0_B,m0_out;
          wire [3:0] m1_A,m1_B,m1_out;
          wire [3:0] m2_out,inv_out;
          wire [3:0] u,c;
          wire [3:0] inv_out1;

          reg [7:0] vr;
          wire [7:0] uvout;

          always @(plain[7:0] or inv) begin
                    if (inv)  begin//means decode
                              vr[7]<=plain[6]^plain[4]^plain[1];
                              vr[6]<=plain[5]^plain[3]^plain[0];
                              vr[5]<=plain[7]^plain[4]^plain[2];
                              vr[4]<=plain[6]^plain[3]^plain[1];
                              vr[3]<=plain[5]^plain[2]^plain[0];
                              vr[2]<=plain[7]^plain[4]^plain[1]^1;
                              vr[1]<=plain[6]^plain[3]^plain[0];
                              vr[0]<=plain[7]^plain[5]^plain[2]^1;
                    end else vr <=plain;
          end




    assign uvout=uv;

    assign      affin_out[7] =  uvout[7] ^  uvout[6] ^  uvout[5] ^  uvout[4] ^  uvout[3] ;
    assign            affin_out[6] =  uvout[6] ^  uvout[5] ^  uvout[4] ^  uvout[3] ^  uvout[2] ^ 1;
    assign            affin_out[5] =  uvout[5] ^  uvout[4] ^  uvout[3] ^  uvout[2] ^  uvout[1] ^ 1;
    assign            affin_out[4] =  uvout[4] ^  uvout[3] ^  uvout[2] ^  uvout[1] ^  uvout[0] ;
    assign            affin_out[3] =  uvout[7] ^  uvout[3] ^  uvout[2] ^  uvout[1] ^  uvout[0] ;
    assign            affin_out[2] =  uvout[7] ^  uvout[6] ^  uvout[2] ^  uvout[1] ^  uvout[0] ;
    assign            affin_out[1] =  uvout[7] ^  uvout[6] ^  uvout[5] ^  uvout[1] ^  uvout[0] ^ 1;
    assign            affin_out[0] =  uvout[7] ^  uvout[6] ^  uvout[5] ^  uvout[4] ^  uvout[0] ^ 1;

          function [3:0] inv16;
```

```
		input [3:0] address;


		case (address)
			0: inv16=4'h0;
			1: inv16=4'h1;
			2: inv16=4'h9;
			3: inv16=4'he;
			4: inv16=4'hd;
			5: inv16=4'hb;
			6: inv16=4'h7;
			7: inv16=4'h6;
			8: inv16=4'hf;
			9: inv16=4'h2;
			10:inv16=4'hc;
			11:inv16=4'h5;
			12:inv16=4'ha;
			13:inv16=4'h4;
			14:inv16=4'h3;
			15:inv16=4'h8;
		endcase

	endfunction

	assign yout=!inv ? affin_out : uvout;


//assign inv_out=inv16(adder2[3:0]);
assign inv_out1=inv16(m2_out[3:0]);
//assign inv_out=inv16(pipe_reg0[3:0]);
assign inv_out=inv_out1;

//変数  AXB  掛け算入力
//下左
assign m0_A[3:0]=inv_out;
//assign m0_B[3:0]=u16c[7:4];
assign m0_B[3:0]=u16c[7:4];

//下右
assign m1_A[3:0]=inv_out;
//assign m1_B[3:0]=adder1[3:0];
assign m1_B[3:0]=adder1[3:0];

assign adder1[3:0]=u16c[3:0] ^ u16c[7:4];

assign u[3:0]=u16c[7:4];
assign c[3:0]=u16c[3:0];


// Add Adder
assign m2_out[0]=u[0]       ^c[0]^ c[2]^ c[0]& u[0] ^ c[3] & u[1] ^ c[2] &u[2] ^ c[1] & u[3] ;
assign m2_out[1]=u[1]^u[3]   ^c[2]^ c[1] & u[0] ^ c[0] & u[1] ^ c[3]& u[1] ^ c[2] &u[2] ^ c[3] & u[2] ^ c[1]& u[3] ^ c
[2]& u[3];
assign m2_out[2]=u[3]       ^c[1]^ c[3]^ c[2] & u[0] ^ c[1] & u[1] ^ c[0] & u[2] ^ c[3] & u[2] ^ c[2]& u[3] ^ c[3] &u
[3];
assign m2_out[3]=u[0]^u[2]   ^c[3]^ c[3] & u[0] ^ c[2] & u[1] ^ c[1]& u[2] ^ c[0]& u[3] ^ c[3]& u[3];


gmul16  mul0(m0_A,m0_B,m0_out);
gmul16  mul1(m1_A,m1_B,m1_out);


assign cu16={m0_out[3:0],m1_out[3:0]};
/* 2^4^2-> 2^8
```

```
    1   0   1   0   0   1   1   1
    0   0   0   0   1   0   0   1
    0   0   1   0   1   0   0   0
    0   0   1   0   1   1   0   0
    0   0   1   1   0   0   0   0
    0   1   0   1   0   0   1   0
    0   1   1   0   1   0   0   0
    0   1   0   1   0   0   1   1
*/
assign uv[0]= cu16[0]  ^    cu16[2]                              ^ cu16[5] ^ cu16[6]   ^  cu16[7];
assign uv[1]=                                  cu16[4] ^                          cu16[7];
assign uv[2]=                  cu16[2]         ^ cu16[4]                    ;
assign uv[3]=                  cu16[2]         ^ cu16[4] ^ cu16[5]                ;
assign uv[4]=                 ^ cu16[2] ^ cu16[3]                                          ;
assign uv[5]=     cu16[1]      ^ cu16[3]                          ^ cu16[6]              ;
assign uv[6]=     cu16[1] ^ cu16[2]         ^ cu16[4]                          ;
assign uv[7]=     cu16[1]      ^ cu16[3]                          ^ cu16[6]   ^  cu16[7] ;

/*
XInv =27   2^8-> 2^4^2

    1   0   0   1   1   0   1   1  h0
    0   0   1   0   0   0   1   0  h1
    0   1   1   0   0   1   0   1  h2
    0   1   1   0   1   1   0   1  h3
    0   1   0   0   0   1   0   1  h4
    0   0   1   1   0   0   0   0  h5
    0   1   0   0   1   0   1   1  h6
    0   0   0   0   0   1   0   1  h7
*/
assign u16c[0]= vr[0]  ^             vr[3] ^ vr[4]   ^     vr[6]   ^ vr[7];
assign u16c[1]=            vr[2]  ^                  vr[6]          ;
assign u16c[2]=      vr[1] ^ vr[2]          ^        vr[5]        ^ vr[7];
assign u16c[3]=      vr[1] ^ vr[2]          ^ vr[4] ^ vr[5]        ^ vr[7];
assign u16c[4]=     vr[1]                           ^ vr[5]        ^ vr[7];
assign u16c[5]=                 vr[2]  ^ vr[3]                                          ;
assign u16c[6]=     vr[1]                  ^ vr[4]       ^vr[6]   ^ vr[7];
assign u16c[7]=                                              vr[5]        ^ vr[7];



endmodule

module gmul16(u,v,uv);
          input [3:0] u,v;
          output [3:0] uv;

 assign uv[0]=v[0] & u[0] ^ v[3] & u[1] ^ v[2] & u[2] ^ v[1] & u[3] ;
 assign uv[1]=v[1] & u[0] ^ v[0] & u[1] ^ v[3] & u[1] ^ v[2] & u[2] ^ v[3] & u[2] ^ v[1] & u[3] ^ v[2] & u[3];
 assign uv[2]=v[2] & u[0] ^ v[1] & u[1] ^ v[0] & u[2] ^ v[3] & u[2] ^ v[2] & u[3] ^ v[3] & u[3];
 assign uv[3]=v[3] & u[0] ^ v[2] & u[1] ^ v[1] & u[2] ^ v[0] & u[3] ^ v[3] & u[3];


endmodule
```

AES 回路（論理合成対象）

```
///   AES CIPHER CORE for Altera FPGA CYCLONE
///   This is derivative work from original code of Rudolf Usselmann.
///   Almost all codes and Interface are changed from the original.
///
///   Features
///     .Specific Core Altera CYCLONE
///     .Encode/Decode with one core
///     .5 PipeLine
///     .for 128bits keys only
///
///
///    Synthesis Result using EP1C4F400C6   Condition (Quartus2 SP2)
///     .212MHz (Excluding IO Interface)
///     .2515CELL
///
///    Copyright (C) 2003 Tak.Sugawara
///    Nov.7.2003
///    Author: Tak.Sugawara
///    www.sugawara-systems.com
///    tech-support@sugawara-systems.com
///




//

//////////////////////////////////////////////////////////////////////
////                                                              ////
////  AES Key Expand Block (for 128 bit keys)                     ////
////                                                              ////
////                                                              ////
////  Author: Rudolf Usselmann                                    ////
////          rudi@asics.ws                                       ////
////                                                              ////
////                                                              ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/   ////
////                                                              ////
//////////////////////////////////////////////////////////////////////
////                                                              ////
//// Copyright (C) 2000-2002 Rudolf Usselmann                     ////
////                         www.asics.ws                         ////
////                         rudi@asics.ws                        ////
////                                                              ////
//// This source file may be used and distributed without         ////
//// restriction provided that this copyright statement is not    ////
//// removed from the file and that any derivative work contains  ////
//// the original copyright notice and the associated disclaimer.////
////                                                              ////
////     THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY      ////
//// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED    ////
//// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS    ////
//// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR       ////
//// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,          ////
//// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES     ////
//// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE    ////
//// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR         ////
//// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF   ////
//// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT   ////
//// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT   ////
//// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE          ////
//// POSSIBILITY OF SUCH DAMAGE.                                  ////
```

```
////                                                         ////
//////////////////////////////////////////////////////////////////////

//  CVS Log
//
//  $Id: aes_key_expand_128.v,v 1.1.1.1 2002/11/09 11:22:38 rudi Exp $
//
//  $Date: 2002/11/09 11:22:38 $
//  $Revision: 1.1.1.1 $
//  $Author: rudi $
//  $Locker:  $
//  $State: Exp $
//
// Change History:
//               $Log: aes_key_expand_128.v,v $
//               Revision 1.1.1.1  2002/11/09 11:22:38  rudi
//               Initial Checkin
//
//
//
//
//
//
`define CYCLONE
module aes_cipher_top(clk, rst, ld_in, done, text_in,
text_out_port,inv_in,kld_in,start_pulse_in,read_enable,kdone,almost_full,clock_out);
input inv_in;
input                   clk, rst;
input                   ld_in;
output                  done;
input      [127:0]  text_in;
output     [127:0]  text_out_port ;
output read_enable;
output kdone;
output almost_full;
output clock_out;
input kld_in;
input start_pulse_in;

reg inv0,inv_top,ld,kld,start_pulse;

reg [127:0] final_text_out01,final_text_out23,final_text_out45,final_text_out6,final_text_out;
wire [127:0] fifo_read_port;

reg [4:0] address;
reg [5:0] counter;
reg read_enable,read_enable_d;
            wire [5:0] usedw;
            wire full,empty;
            wire fifo_write,fifo_read;
            wire sclr;
reg [3:0] state;
reg load_d;
reg almost_full;
wire over5,go,loadable;
wire load;

fifo64 fifo(
            .data(text_in),
            .wrreq(ld_in),
            .rdreq(load),
            .clock(clk),
            .sclr(0),
            .q(fifo_read_port),
```

```
        .full(full),
        .empty(empty),
        .usedw(usedw));

        assign load=state[0] | state[1];
        assign over5=usedw >=5;
        assign go=over5 && loadable;
        assign clock_out=clk;
wire [127:0] sr_port0,ark_port0;
wire [127:0] final_out;
wire read_enable0;
wire kdone0;
wire done0;
        always @(posedge clk or negedge rst) begin
                if (!rst) almost_full<=0;
                else  almost_full <=usedw>=60;
        end

        always @(posedge clk or negedge rst) begin
                if (!rst) state<=0;
                else begin
                        case (state)
                                4'b0000:if (go )   state  <=4'b0001;
                                4'b0001:                  state  <=4'b0011;
                                4'b0011:             state  <=4'b0111;
                                4'b0111:             state  <=4'b1111;
                                4'b1111:                 state  <=4'b1110;
                                4'b1110:                 state<=4'b1100;
                                4'b1100:             state<=4'b0000;//
                                4'b1000:             state<=4'b0000;
                                default: state <=4'b0000;
                        endcase
                end
        end
        always @(posedge clk) begin
                inv0 <=inv_in;
                inv_top<=inv_in;
                start_pulse<=start_pulse_in;
                kld<=kld_in;
                ld<=ld_in;
                load_d<=load;
                //text_in_reg <=text_in;
        end



        always @(posedge clk or negedge rst) begin//
         if(!rst)   read_enable <=0;
         else                read_enable <=read_enable0;
        end







        assign kdone=kdone0;
        assign done=done0;


        assign text_out_port=read_enable ? final_out : 128'hzzzz_zzzz_zzzz_zzzz_zzzz_zzzz_zzzz_zzzz_zzzz;
```

```
 aes_cipher_module aes0(clk, rst,load_d,
done0,fifo_read_port,final_out,inv0,kld_in,start_pulse,read_enable0,kdone0,loadable,text_in);

endmodule
module aes_cipher_module(clk, rst, ld, done, text_in,
final_out,inv,kld,start_pulse,read_enable,kdone,loadable,key_text);

input inv;
input           clk, rst;
input           ld;
output          done;
input   [127:0] text_in,key_text;
output  [127:0] final_out ;
output read_enable;
output kdone;
input kld;
input start_pulse;
output loadable;
////////////////////////////////////////////////////////////////////
//
// Local Wires
//

wire    [31:0]  w0, w1, w2, w3;

wire    [7:0]   sa00, sa01, sa02, sa03;
wire    [7:0]   sa10, sa11, sa12, sa13;
wire    [7:0]   sa20, sa21, sa22, sa23;
wire    [7:0]   sa30, sa31, sa32, sa33;



wire    [7:0]   sa00_next, sa01_next, sa02_next, sa03_next;
wire    [7:0]   sa10_next, sa11_next, sa12_next, sa13_next;
wire    [7:0]   sa20_next, sa21_next, sa22_next, sa23_next;
wire    [7:0]   sa30_next, sa31_next, sa32_next, sa33_next;

wire    [7:0]   sa00_next_inv, sa01_next_inv, sa02_next_inv, sa03_next_inv;
wire    [7:0]   sa10_next_inv, sa11_next_inv, sa12_next_inv, sa13_next_inv;
wire    [7:0]   sa20_next_inv, sa21_next_inv, sa22_next_inv, sa23_next_inv;
wire    [7:0]   sa30_next_inv, sa31_next_inv, sa32_next_inv, sa33_next_inv;



wire    [7:0]   sa00_sub, sa01_sub, sa02_sub, sa03_sub;
wire    [7:0]   sa10_sub, sa11_sub, sa12_sub, sa13_sub;
wire    [7:0]   sa20_sub, sa21_sub, sa22_sub, sa23_sub;
wire    [7:0]   sa30_sub, sa31_sub, sa32_sub, sa33_sub;
wire    [7:0]   sa00_sr, sa01_sr, sa02_sr, sa03_sr;
wire    [7:0]   sa10_sr, sa11_sr, sa12_sr, sa13_sr;
wire    [7:0]   sa20_sr, sa21_sr, sa22_sr, sa23_sr;
wire    [7:0]   sa30_sr, sa31_sr, sa32_sr, sa33_sr;



wire    [7:0]   sa00_sr_inv, sa01_sr_inv, sa02_sr_inv, sa03_sr_inv;
wire    [7:0]   sa10_sr_inv, sa11_sr_inv, sa12_sr_inv, sa13_sr_inv;
wire    [7:0]   sa20_sr_inv, sa21_sr_inv, sa22_sr_inv, sa23_sr_inv;
wire    [7:0]   sa30_sr_inv, sa31_sr_inv, sa32_sr_inv, sa33_sr_inv;

wire    [7:0]   sa00_mc, sa01_mc, sa02_mc, sa03_mc;
wire    [7:0]   sa10_mc, sa11_mc, sa12_mc, sa13_mc;
wire    [7:0]   sa20_mc, sa21_mc, sa22_mc, sa23_mc;
```

```
wire	[7:0]	sa30_mc, sa31_mc, sa32_mc, sa33_mc;

wire	[7:0]	sa00_ark, sa01_ark, sa02_ark, sa03_ark;
wire	[7:0]	sa10_ark, sa11_ark, sa12_ark, sa13_ark;
wire	[7:0]	sa20_ark, sa21_ark, sa22_ark, sa23_ark;
wire	[7:0]	sa30_ark, sa31_ark, sa32_ark, sa33_ark;

reg		done;
reg		kdone;
reg   [3:0] pass_counter;
reg   [2:0] usedw_counter;
wire   excuting;
reg   d0,d1,d2,d3,d4;
reg   [3:0] state;
reg   [127:0] pipe_final,data_set_reg,final_out;
reg  read_enable;
 wire [127:0] fifo_read_port;
 wire [127:0] shift_out,ark_port,sr_port,wport,next_port,next_inv_port,text_port;
 wire [127:0] final_key;
 wire  sr_port_sel=!inv;
 reg next_inv_sel,next_port_sel;
 reg ld_d;




	always @(posedge clk) begin
		if ( next_inv_sel ) pipe_final <=next_inv_port;
		else if (next_port_sel) pipe_final <=next_port;
	end

	always @(posedge clk) begin
		if (ld ) data_set_reg <=text_in^wport;
		else     data_set_reg <=pipe_final^wport;
	end
	always @(posedge clk) begin
		if( inv)  final_out <=ark_port^final_key;
		else          final_out <=sr_port^final_key;
	end


	always @(posedge clk or negedge rst) begin
		if (!rst) read_enable<=0;
		else   if (pass_counter==10 && d2 )
				read_enable <=1;

		else   if (read_enable && d2)
				read_enable <=0;
	end




	always @(posedge clk or negedge rst) begin
		if (!rst) d0<=0;
		else  if (state==4'b0000  && ld ) d0 <=1;
		else  if (state==4'b0000)   d0<=0;
		else  if ( state !=4'b0000) d0 <=d4;
	end
```

```
always @(posedge clk or negedge rst) begin
        if (!rst) d1<=0;
        else d1<=d0;
end
always @(posedge clk or negedge rst) begin
        if (!rst) d2<=0;
        else d2<=d1;
end
always @(posedge clk or negedge rst) begin
        if (!rst) d3<=0;
        else d3<=d2;
end
always @(posedge clk or negedge rst) begin
        if (!rst) d4<=0;
        else d4<=d3;
end
always @(posedge clk or negedge rst) begin
        if (!rst) pass_counter<=0;
        else if (state==4'b0000 ) pass_counter<=1;
        else if (state==4'b1110 &&  d4 && ld) pass_counter<=1;
        else if (d3 ) pass_counter<=pass_counter+1;
end



always @(posedge clk or negedge rst) begin
        if (!rst) state<=0;
        else begin
                case (state)
                        4'b0000:if (ld)     state  <=4'b0001;
                        4'b0001:if (d2)     state  <=4'b0011;
                        4'b0011:if (d3)       state  <=4'b0111;
                        4'b0111:if (pass_counter==10 && d3)     state  <=4'b1110;
                        4'b1111:if (d3)     state  <=4'b0111;
                        4'b1110: if (d4 && ld) state<=4'b0001;
                                else if (d1) state<=4'b1100;
                        4'b1100:            state<=4'b1000;//done delay
                        4'b1000:            state<=4'b0000;
                        default: state <=4'b0000;
                endcase
        end
end
always @(posedge clk or negedge rst) begin
        if (!rst) done<=1;
        else if (ld) done<=0;
        else begin
                if(  !(ld  | excuting | read_enable) ) done<=1;
        end
end


always @(posedge clk or negedge rst ) begin
        if (!rst) next_port_sel<=0;
        else if( ((state==4'b0001 ) || (state==4'b1111 ))  && d2 && !inv)      next_port_sel<=1;//
        else if(pass_counter==10 && d2 )    next_port_sel<=0;//
end
always @(posedge clk or negedge rst ) begin
        if (!rst)  next_inv_sel<=0;
        else if(((state==4'b0001 ) || (state==4'b1111 ))  && d2 &&inv)  next_inv_sel<=1;//
        else if(pass_counter==10 && d2)  next_inv_sel<=0;//
end
always @(posedge clk or negedge rst ) begin
        if (!rst)  ld_d<=0;
        else       ld_d<=ld;//
```

```
        end


        assign excuting =state[0] | state[1] | state[2] | state[3];
        assign loadable= !excuting || (state==4'b0111 && pass_counter==10 && d2);

////////////////////////////////////////////////////////////////
//
// Misc Logic
//




////////////////////////////////////////////////////////////////
//
// Initial Permutation (AddRoundKey)
//



assign      {sa00,sa10,sa20,sa30,
             sa01,sa11,sa21,sa31,
             sa02,sa12,sa22,sa32,
             sa03,sa13,sa23,sa33}=data_set_reg;

        assign ark_port={sa00_ark,sa10_ark,sa20_ark,sa30_ark,
                         sa01_ark,sa11_ark,sa21_ark,sa31_ark,
                         sa02_ark,sa12_ark,sa22_ark,sa32_ark,
                         sa03_ark,sa13_ark,sa23_ark,sa33_ark};

        assign sr_port={ sa00_sr,sa10_sr,sa20_sr,sa30_sr,
                         sa01_sr,sa11_sr,sa21_sr,sa31_sr,
                         sa02_sr,sa12_sr,sa22_sr,sa32_sr,
                         sa03_sr,sa13_sr,sa23_sr,sa33_sr};

      assign next_port={sa00_next,sa10_next,sa20_next,sa30_next,
                         sa01_next,sa11_next,sa21_next,sa31_next,
                         sa02_next,sa12_next,sa22_next,sa32_next,
                         sa03_next,sa13_next,sa23_next,sa33_next};

      assign next_inv_port={sa00_next_inv,sa10_next_inv,sa20_next_inv,sa30_next_inv,
                            sa01_next_inv,sa11_next_inv,sa21_next_inv,sa31_next_inv,
                            sa02_next_inv,sa12_next_inv,sa22_next_inv,sa32_next_inv,
                            sa03_next_inv,sa13_next_inv,sa23_next_inv,sa33_next_inv};

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




////////////////////////////////////////////////////////////////
//
// Round Permutations
//


assign sa00_sr_inv =sa00;
assign sa01_sr_inv =sa01;
```

```
assign sa02_sr_inv =sa02;
assign sa03_sr_inv =sa03;
assign sa10_sr_inv =sa13;
assign sa11_sr_inv =sa10;
assign sa12_sr_inv =sa11;
assign sa13_sr_inv =sa12;
assign sa20_sr_inv =sa22;
assign sa21_sr_inv =sa23;
assign sa22_sr_inv =sa20;
assign sa23_sr_inv =sa21;
assign sa30_sr_inv =sa31;
assign sa31_sr_inv =sa32;
assign sa32_sr_inv =sa33;
assign sa33_sr_inv =sa30;

assign sa00_sr =sa00_sub;
assign sa01_sr =sa01_sub;
assign sa02_sr =sa02_sub;
assign sa03_sr =sa03_sub;
assign sa10_sr =sa11_sub;
assign sa11_sr =sa12_sub;
assign sa12_sr =sa13_sub;
assign sa13_sr =sa10_sub;
assign sa20_sr =sa22_sub;
assign sa21_sr =sa23_sub;
assign sa22_sr =sa20_sub;
assign sa23_sr =sa21_sub;
assign sa30_sr =sa33_sub;
assign sa31_sr =sa30_sub;
assign sa32_sr =sa31_sub;
assign sa33_sr =sa32_sub;



assign {sa00_mc, sa10_mc, sa20_mc, sa30_mc}  = mix_col(sa00_sr,sa10_sr,sa20_sr,sa30_sr);
assign {sa01_mc, sa11_mc, sa21_mc, sa31_mc}  = mix_col(sa01_sr,sa11_sr,sa21_sr,sa31_sr);
assign {sa02_mc, sa12_mc, sa22_mc, sa32_mc}  = mix_col(sa02_sr,sa12_sr,sa22_sr,sa32_sr);
assign {sa03_mc, sa13_mc, sa23_mc, sa33_mc}  = mix_col(sa03_sr,sa13_sr,sa23_sr,sa33_sr);

assign sa00_next = sa00_mc;
assign sa01_next = sa01_mc;
assign sa02_next = sa02_mc;
assign sa03_next = sa03_mc;
assign sa10_next = sa10_mc;
assign sa11_next = sa11_mc;
assign sa12_next = sa12_mc;
assign sa13_next = sa13_mc;
assign sa20_next = sa20_mc;
assign sa21_next = sa21_mc;
assign sa22_next = sa22_mc;
assign sa23_next = sa23_mc;
assign sa30_next = sa30_mc;
assign sa31_next = sa31_mc;
assign sa32_next = sa32_mc;
assign sa33_next = sa33_mc;



assign shift_out[007:000]=sr_port_sel ?sa33_sr : sa33_ark;
assign shift_out[015:008]=sr_port_sel ?sa23_sr : sa23_ark;
assign shift_out[023:016]=sr_port_sel ?sa13_sr : sa13_ark;
assign shift_out[031:024]=sr_port_sel ?sa03_sr : sa03_ark;
```

```
assign shift_out[039:032]=sr_port_sel ?sa32_sr : sa32_ark;
assign shift_out[047:040]=sr_port_sel ?sa22_sr : sa22_ark;
assign shift_out[055:048]=sr_port_sel ?sa12_sr : sa12_ark;
assign shift_out[063:056]=sr_port_sel ?sa02_sr : sa02_ark;
assign shift_out[071:064]=sr_port_sel ?sa31_sr : sa31_ark;
assign shift_out[079:072]=sr_port_sel ?sa21_sr : sa21_ark;
assign shift_out[087:080]=sr_port_sel ?sa11_sr : sa11_ark;
assign shift_out[095:088]=sr_port_sel ?sa01_sr : sa01_ark;
assign shift_out[103:096]=sr_port_sel ?sa30_sr : sa30_ark;
assign shift_out[111:104]=sr_port_sel ?sa20_sr : sa20_ark;
assign shift_out[119:112]=sr_port_sel ?sa10_sr : sa10_ark;
assign shift_out[127:120]=sr_port_sel ?sa00_sr : sa00_ark;



assign {sa00_next_inv, sa10_next_inv, sa20_next_inv, sa30_next_inv} = inv_mix_col
(sa00_ark,sa10_ark,sa20_ark,sa30_ark);
assign {sa01_next_inv, sa11_next_inv, sa21_next_inv, sa31_next_inv} = inv_mix_col
(sa01_ark,sa11_ark,sa21_ark,sa31_ark);
assign {sa02_next_inv, sa12_next_inv, sa22_next_inv, sa32_next_inv} = inv_mix_col
(sa02_ark,sa12_ark,sa22_ark,sa32_ark);
assign {sa03_next_inv, sa13_next_inv, sa23_next_inv, sa33_next_inv} = inv_mix_col
(sa03_ark,sa13_ark,sa23_ark,sa33_ark);



//
////////////////////////////////////////////////////////////////////
//
// Inverse Generic Functions
//

function [31:0] inv_mix_col;
input     [7:0]      s0,s1,s2,s3;
begin
inv_mix_col[31:24]=pmul_e(s0)^pmul_b(s1)^pmul_d(s2)^pmul_9(s3);
inv_mix_col[23:16]=pmul_9(s0)^pmul_e(s1)^pmul_b(s2)^pmul_d(s3);
inv_mix_col[15:08]=pmul_d(s0)^pmul_9(s1)^pmul_e(s2)^pmul_b(s3);
inv_mix_col[07:00]=pmul_b(s0)^pmul_d(s1)^pmul_9(s2)^pmul_e(s3);
end
endfunction

// Some synthesis tools don't like xtime being called recursevly ...
function [7:0] pmul_e;
input [7:0] b;
reg [7:0] two,four,eight;
reg [7:0] temp;
begin
   two=xtime(b);four=xtime(two);eight=xtime(four);pmul_e=eight^four^two;
end
endfunction

function [7:0] pmul_9;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_9=eight^b;

end
endfunction

function [7:0] pmul_d;
```

```
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_d=eight^four^b;
end
endfunction

function [7:0] pmul_b;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_b=eight^two^b;
end
endfunction

function [7:0] xtime;
input [7:0] b;xtime={b[6:0],1'b0}^(8'h1b&{8{b[7]}});
endfunction

//////////////////////////////////////////////////////////////////////
//
// Key Buffer
//

reg         [4:0]      kcnt;
wire [31:0] wk3,wk2,wk1,wk0;
wire [127:0] ram_out_data,ram_in_data;


always @(posedge clk or negedge rst)//
            if(!rst)    kcnt <=   5'h00;
            else  if(kld)          kcnt   <=   kcnt + 5'h01;
            else  if(start_pulse && !inv) kcnt <=1;//for encode
            else  if(start_pulse &&  inv) kcnt <=17;//for decode
            else  if(d1 && kcnt==5'h0a && !inv ) kcnt <=1;
            else  if(d1 && kcnt==5'h1a && inv  ) kcnt <=17;
            else  if(d1)                        kcnt<=kcnt+1;

always @(posedge clk or negedge rst)
            if (!rst) kdone <=1;
            else if ((kld || kcnt !=5'h1c) && state !=4'b0000) kdone<=0;
                else if(kcnt==5'h1c)     kdone<=1;

assign ram_in_data= key_text;
wire [4:0] address,final_key_address;
    assign address= kcnt;
`ifdef CYCLONE
            wire [31:0] wr3,wr2,wr1,wr0;
        assign w0=wr0;
            assign w1=wr1;
            assign w2=wr2;
            assign w3=wr3;
            assign {wr0,wr1,wr2,wr3} =ram_out_data;
    cram ram1(.address(address),.clock(clk),.data(ram_in_data),.wren(kld),.q(ram_out_data));
    cram ram2(.address(kld? address :final_key_address),.clock(clk),.data(ram_in_data),.wren(kld),.q(final_key));

      assign final_key_address=!inv ? 5'h0b : 5'h1b;


`else

`endif
```

```
        assign wk3=w3;
        assign wk2=w2;
        assign wk1=w1;
        assign wk0=w0;

////////////////////////////////////////////////////////////////////
//
// Generic Functions
//

function [31:0] mix_col;
input   [7:0]   s0,s1,s2,s3;
reg     [7:0]   s0_o,s1_o,s2_o,s3_o;
begin
mix_col[31:24]=xtime(s0)^xtime(s1)^s1^s2^s3;
mix_col[23:16]=s0^xtime(s1)^xtime(s2)^s2^s3;
mix_col[15:08]=s0^s1^xtime(s2)^xtime(s3)^s3;
mix_col[07:00]=xtime(s0)^s0^s1^s2^xtime(s3);
end
endfunction

//function [7:0] xtime;
//input [7:0] b; xtime={b[6:0],1'b0}^(8'h1b&{8{b[7]}});
//endfunction


wire [7:0] gi00=sa00;
wire [7:0] gi01=sa01;
wire [7:0] gi02=sa02;
wire [7:0] gi03=sa03;
wire [7:0] gi10=sa10;
wire [7:0] gi11=sa11;
wire [7:0] gi12=sa12;
wire [7:0] gi13=sa13;
wire [7:0] gi20=sa20;
wire [7:0] gi21=sa21;
wire [7:0] gi22=sa22;
wire [7:0] gi23=sa23;
wire [7:0] gi30=sa30;
wire [7:0] gi31=sa31;
wire [7:0] gi32=sa32;
wire [7:0] gi33=sa33;


`define PIPE

`ifdef PIPE
//clock,plain,yout,inv,reset,f_ark
        gmul_pipe_ver2  us00(.clock(clk),.plain(gi00),.plain_inv(gi00),.yout(sa00_sub),.inv(inv),.reset(!rst),.f_ark
(sa00_ark));
        gmul_pipe_ver2  us01(.clock(clk),.plain(gi01),.plain_inv(gi01),.yout(sa01_sub),.inv(inv),.reset(!rst),.f_ark
(sa01_ark));
        gmul_pipe_ver2  us02(.clock(clk),.plain(gi02),.plain_inv(gi02),.yout(sa02_sub),.inv(inv),.reset(!rst),.f_ark
(sa02_ark));
        gmul_pipe_ver2  us03(.clock(clk),.plain(gi03),.plain_inv(gi03),.yout(sa03_sub),.inv(inv),.reset(!rst),.f_ark
(sa03_ark));
        gmul_pipe_ver2  us10(.clock(clk),.plain(gi10),.plain_inv(sa10_sr_inv),.yout(sa10_sub),.inv(inv),.reset(!rst),.
f_ark(sa10_ark));
        gmul_pipe_ver2  us11(.clock(clk),.plain(gi11),.plain_inv(sa11_sr_inv),.yout(sa11_sub),.inv(inv),.reset(!rst),.
f_ark(sa11_ark));
        gmul_pipe_ver2  us12(.clock(clk),.plain(gi12),.plain_inv(sa12_sr_inv),.yout(sa12_sub),.inv(inv),.reset(!rst),.
f_ark(sa12_ark));
        gmul_pipe_ver2  us13(.clock(clk),.plain(gi13),.plain_inv(sa13_sr_inv),.yout(sa13_sub),.inv(inv),.reset(!rst),.
```

```
f_ark(sa13_ark));
          gmul_pipe_ver2  us20(.clock(clk),.plain(gi20),.plain_inv(sa20_sr_inv),.yout(sa20_sub),.inv(inv),.reset(!rst),.
f_ark(sa20_ark));
        gmul_pipe_ver2  us21(.clock(clk),.plain(gi21),.plain_inv(sa21_sr_inv),.yout(sa21_sub),.inv(inv),.reset(!rst),.
f_ark(sa21_ark));
        gmul_pipe_ver2  us22(.clock(clk),.plain(gi22),.plain_inv(sa22_sr_inv),.yout(sa22_sub),.inv(inv),.reset(!rst),.
f_ark(sa22_ark));
        gmul_pipe_ver2  us23(.clock(clk),.plain(gi23),.plain_inv(sa23_sr_inv),.yout(sa23_sub),.inv(inv),.reset(!rst),.
f_ark(sa23_ark));
        gmul_pipe_ver2  us30(.clock(clk),.plain(gi30),.plain_inv(sa30_sr_inv),.yout(sa30_sub),.inv(inv),.reset(!rst),.
f_ark(sa30_ark));
        gmul_pipe_ver2  us31(.clock(clk),.plain(gi31),.plain_inv(sa31_sr_inv),.yout(sa31_sub),.inv(inv),.reset(!rst),.
f_ark(sa31_ark));
        gmul_pipe_ver2  us32(.clock(clk),.plain(gi32),.plain_inv(sa32_sr_inv),.yout(sa32_sub),.inv(inv),.reset(!rst),.
f_ark(sa32_ark));
        gmul_pipe_ver2  us33(.clock(clk),.plain(gi33),.plain_inv(sa33_sr_inv),.yout(sa33_sub),.inv(inv),.reset(!rst),.
f_ark(sa33_ark));


`else


`endif

function [127:0] mux4;
          input [127:0] a;
          input [127:0] b;
          input [127:0] c;
          input [127:0] d;
          input [1:0] sel;

          case (sel)
                    2'b00: mux4=a;
                    2'b01: mux4=b;
                    2'b10: mux4=c;
                    2'b11: mux4=d;
          endcase
endfunction

function [127:0] mux2;
          input [127:0] a;
          input [127:0] b;

          input sel;

          case (sel)
                    0: mux2=a;
                    1: mux2=b;
          endcase
endfunction

endmodule



//subbytes/invsubbytes 回路
module gmul_pipe_ver2(clock,plain,plain_inv,yout,inv,reset,f_ark);
          input clock;
          input inv;
          input [7:0] plain,plain_inv;
          input reset;
          output [7:0] yout,f_ark;
        wire [7:0] uv,affin_out;
```

```
	wire [7:0] u16,u16c, cu16;
	wire [3:0] pow2,adder1,adder2,x14;
	wire [3:0] m0_A,m0_B,m0_out;
	wire [3:0] m1_A,m1_B,m1_out;
	wire [3:0] m2_out,inv_out;
	wire [3:0] u,c;
	wire [3:0] inv_out1;

	reg [7:0] vr;
	reg [7:0] uvout,f_ark;
	reg [3:0] pipe_reg0,pipe_reg1,pipe_reg2;
	wire [7:0] vr_in;
	reg inv_reg;
	always @(posedge clock or posedge reset) begin
		if (reset) vr<=0;
		else vr <=u16c;

	end


	always @(posedge clock) begin
		inv_reg <=inv;
	end



always @(posedge clock or posedge reset) begin
		if (reset) uvout<=0;
		else begin
			uvout <=affin_out;
		end
end

always @(posedge clock or posedge reset) begin
		if (reset) f_ark<=0;
		else begin
			 f_ark <=uv;
		end
end




	function [3:0] inv16;
		input [3:0] address;


		case (address)
			0: inv16=4'h0;
			1: inv16=4'h1;
			2: inv16=4'h9;
			3: inv16=4'he;
			4: inv16=4'hd;
			5: inv16=4'hb;
			6: inv16=4'h7;
			7: inv16=4'h6;
			8: inv16=4'hf;
			9: inv16=4'h2;
			10:inv16=4'hc;
			11:inv16=4'h5;
			12:inv16=4'ha;
			13:inv16=4'h4;
			14:inv16=4'h3;
			15:inv16=4'h8;
```

```
                endcase

        endfunction

        assign yout=uvout;

 assign      vr_in[7]=!inv_reg? plain[7] :plain_inv[6]^plain_inv[4]^plain_inv[1];
 assign      vr_in[6]=!inv_reg? plain[6] :plain_inv[5]^plain_inv[3]^plain_inv[0];
 assign      vr_in[5]=!inv_reg? plain[5] :plain_inv[7]^plain_inv[4]^plain_inv[2];
 assign      vr_in[4]=!inv_reg? plain[4] :plain_inv[6]^plain_inv[3]^plain_inv[1];
 assign      vr_in[3]=!inv_reg? plain[3] :plain_inv[5]^plain_inv[2]^plain_inv[0];
 assign      vr_in[2]=!inv_reg? plain[2] :plain_inv[7]^plain_inv[4]^plain_inv[1]^1;
 assign      vr_in[1]=!inv_reg? plain[1] :plain_inv[6]^plain_inv[3]^plain_inv[0];
 assign      vr_in[0]=!inv_reg? plain[0] :plain_inv[7]^plain_inv[5]^plain_inv[2]^1;




assign inv_out1=inv16(m2_out[3:0]);
assign inv_out=pipe_reg0;

assign m0_A[3:0]=inv_out;
assign m0_B[3:0]=pipe_reg2[3:0];

assign m1_A[3:0]=inv_out;
assign m1_B[3:0]=pipe_reg1;

assign adder1[3:0]=vr[3:0] ^ vr[7:4];

assign u[3:0]=vr[7:4];
assign c[3:0]=vr[3:0];


// Add Adder
assign m2_out[0]=u[0]       ^c[0]^ c[2]^ c[0]& u[0] ^ c[3] & u[1] ^ c[2] &u[2] ^ c[1] & u[3] ;
assign m2_out[1]=u[1]^u[3]   ^c[2]^ c[1] & u[0] ^ c[0] & u[1] ^ c[3]& u[1] ^ c[2] &u[2] ^ c[3] & u[2] ^ c[1]& u[3] ^ c
[2]& u[3];
assign m2_out[2]=u[3]       ^c[1]^ c[3]^ c[2] & u[0] ^ c[1] & u[1] ^ c[0] & u[2] ^ c[3] & u[2] ^ c[2]& u[3] ^ c[3] &u
[3];
assign m2_out[3]=u[0]^u[2]   ^c[3]^ c[3] & u[0] ^ c[2] & u[1] ^ c[1]& u[2] ^ c[0]& u[3] ^ c[3]& u[3];

always @(posedge clock or posedge reset) begin
                if (reset) begin
                        pipe_reg0<=0;
                        pipe_reg1<=0;
                        pipe_reg2<=0;
                end
                else begin
                        pipe_reg0 <=inv_out1 ;
                        pipe_reg1 <=adder1[3:0];
                        pipe_reg2 <=vr[7:4];
                end
    end



gmul16  mul0(.u(m0_A),.v(m0_B),.uv(m0_out));
gmul16  mul1(.u(m1_A),.v(m1_B),.uv(m1_out));
```

```
assign cu16={m0_out[3:0],m1_out[3:0]};

/*
X =

UVX={{1,    0,    0,    0,    1,    0,    1,    0},
     {0,    0,    0,    0,    1,    1,    0,    1},
     {0,    1,    0,    0,    1,    1,    1,    0},
     {0,    1,    0,    0,    1,    1,    0,    1},
     {0,    1,    0,    1,    1,    0,    1,    0},
     {0,    0,    1,    0,    0,    1,    0,    1},
     {0,    1,    1,    1,    0,    1,    1,    1},
     {0,    0,    1,    0,    0,    1,    0,    0}};
*/
assign uv[0]= cu16[0]                                ^cu16[4]                  ^cu16[6]       ;
assign uv[1]=                                  cu16[4] ^ cu16[5]       ^cu16[7] ;
assign uv[2]=         cu16[1]                      ^cu16[4] ^ cu16[5]  ^cu16[6]          ;
assign uv[3]=         cu16[1]                      ^cu16[4] ^ cu16[5]          ^cu16[7] ;
assign uv[4]=       cu16[1]     ^cu16[3]^cu16[4]        ^cu16[6]            ;
assign uv[5]=               cu16[2]                  ^      cu16[5]         ^cu16[7] ;
assign uv[6]=       cu16[1] ^ cu16[2]^cu16[3]        ^ cu16[5]  ^cu16[6]^cu16[7] ;
assign uv[7]=               cu16[2]                  ^      cu16[5]                  ;




/*
AffinX =27 max6
AffinX={{1,    0,    1,    0,    0,    1,    1,    0},
        {1,    1,    1,    1,    0,    0,    0,    1},
        {1,    0,    0,    1,    1,    0,    1,    0},
        {1,    0,    1,    0,    0,    0,    0,    0},
        {1,    1,    0,    1,    1,    1,    1,    0},
        {0,    1,    1,    1,    0,    0,    0,    1},
        {0,    0,    0,    0,    1,    0,    1,    1},
        {0,    1,    1,    0,    0,    0,    0,    1}};
*/
assign affin_out[0]= cu16[0]            ^ cu16[2]                          ^cu16[5] ^ cu16[6]                  ^ 1;
assign affin_out[1]= cu16[0]^cu16[1]^ cu16[2] ^cu16[3]                                                    ^ cu16[7]^ 1;
assign affin_out[2]= cu16[0]                      ^cu16[3] ^ cu16[4]        ^ cu16[6]             ;
assign affin_out[3]= cu16[0]            ^ cu16[2]                                                                ;
assign affin_out[4]= cu16[0]^cu16[1]        ^cu16[3] ^ cu16[4] ^cu16[5] ^ cu16[6]                 ;
assign affin_out[5]=        cu16[1]^ cu16[2] ^cu16[3]                          ^ cu16[7]^ 1;
assign affin_out[6]=                                ^ cu16[4]        ^ cu16[6]       ^ cu16[7]^ 1;
assign affin_out[7]=        cu16[1]^ cu16[2]                                                          ^ cu16[7];




/*   XInv =28
     {{1,     0,     1,     1,     1,     0,     1,     1},
      {0,     1,     0,     1,     0,     0,     0,     0},
      {0,     1,     0,     0,     1,     0,     1,     0},
      {0,     1,     1,     0,     0,     0,     1,     1},
      {0,     0,     0,     0,     1,     1,     1,     0},
      {0,     1,     0,     0,     1,     0,     1,     1},
      {0,     0,     1,     1,     0,     1,     0,     1},
      {0,     0,     0,     0,     0,     1,     0,     1}};
*/
assign u16c[0]= vr_in[0]  ^     vr_in[2]  ^ vr_in[3] ^ vr_in[4]   ^      vr_in[6]   ^ vr_in[7];
assign u16c[1]=      vr_in[1] ^          vr_in[3]                                          ;
assign u16c[2]=       vr_in[1] ^                vr_in[4]       ^  vr_in[6]         ;
assign u16c[3]=       vr_in[1] ^ vr_in[2]                          ^vr_in[6]   ^ vr_in[7];
```

```
assign u16c[4]=                                   vr_in[4]  ^ vr_in[5]^vr_in[6]        ;
assign u16c[5]=      vr_in[1]  ^                  ^ vr_in[4]              ^vr_in[6]  ^  vr_in[7];
assign u16c[6]=                  vr_in[2]  ^ vr_in[3]          ^vr_in[5]        ^ vr_in[7];
assign u16c[7]=                                                  vr_in[5]        ^ vr_in[7];



endmodule

module gmul16(u,v,uv);
            input [3:0] u,v;
            output [3:0] uv;

 assign uv[0]=v[0] & u[0] ^ v[3] & u[1] ^ v[2] & u[2] ^ v[1] & u[3] ;
 assign uv[1]=v[1] & u[0] ^ v[0] & u[1] ^ v[3] & u[1] ^ v[2] & u[2] ^ v[3] & u[2] ^ v[1] & u[3] ^ v[2] & u[3];
 assign uv[2]=v[2] & u[0] ^ v[1] & u[1] ^ v[0] & u[2] ^ v[3] & u[2] ^ v[2] & u[3] ^ v[3] & u[3];
 assign uv[3]=v[3] & u[0] ^ v[2] & u[1] ^ v[1] & u[2] ^ v[0] & u[3] ^ v[3] & u[3];


endmodule
```