山东大学
SHANDONG UNIVERSITY

分组密码算法的硬件实现

密码工程

2023 年 11 月 29 日

李 昕 202100460065

21 级密码 2 班

摘　要

SM4 是一种分组密码算法，由我国国家密码管理局在 2012 年发布，常用于无线互联网加密等领域，其分组长度为 128 位（即 16 字节，4 字），密钥长度也为 128 位（即 16 字节，4 字）。加密算法与密钥扩展算法均采用 32 轮非线性迭代结构，以字（32 位）为单位进行加密运算，每一次迭代运算均为一轮变换函数 F。SM4 算法加/解密算法的结构相同，只是使用轮密钥相反，其中解密轮密钥是加密轮密钥的逆序。

SM4 有很高的灵活性，所采用的 S 盒可以灵活地被替换，以应对突发性的安全威胁。算法的 32 轮迭代采用串行处理，这与 AES 中每轮使用代换和混淆并行地处理整个分组有很大不同。

在本次实验中，利用 Verilog 语言实现 SM4 算法的硬件实现。

关键词：SM4 算法　Verilog

目录

1 实验要求

问题一 分析 SM4 的结构和实现细节，利用 Verilog 语言实现 SM4 密码算法的硬件实现，并利用软件工具进行仿真测试。

2 符号标记说明

符号	解释
$\oplus$	异或
S-box	S 盒
$<<< i$	32 位循环左移 i 位
Z_2^n	比特长度为 n 的二进制序列集合

3 实验准备

3.1 SM4 算法的算法过程 [1]

3.1.1 密钥及密钥参量

SM4 分组密码算法的加密密钥长度为 128b，表示为 $MK = (MK0, MK1, MK2, MK3)$，其中 $MK_i(i = 0\ 1, 2, 3)$ 为字。

轮密钥表示为 $(rk_0, rk_1, \cdots, rk_31)$，其中 $rk_i(i = 0, 1, \cdots\ 31)$ 为 32b。轮密钥由加密密钥生成。

$FK = (FK_1, FK_2, FK_3, FK_4)$ 为系统参数，$CK = (CK_0, CK_1, \cdots, CK_31)$ 为固定参数，用于密钥扩展算法，其中 $FK_i(i = 0, 1, \cdots\ 3)$，$CK_i(i = 0, 1, \cdots\ 31)$ 均为 32b。

3.1.2 SM4 轮函数 F 的构造

轮函数 $F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i)$ 接收 5 个 1 字的参数，前 4 个字 $(X_i, X_{i+1}, X_{i+2}, X_{i+3})$ 为明文字或者迭代中间值，最后一个参数为 1 字的轮密钥 rk_i 。轮函数的输出结果为 1 字

轮函数内部需要执行的运算为 $F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i) = X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i)$, 公式后边的 T 为合成置换。

3.1.3 SM4 合成置换 T

合成置换 T 接收 1 字的输入 A ，得出 1 字的输出 C 。它包含非线性变换 τ 和线性变换 L 两个过程，即 $C = T(A) = L(\tau(A))$

(1) 非线性变换 τ

非线性变换接收 1 字（即 4 个字节）的输入，记为 $A = (a_0, a_1, a_2, a_3)$ （其中 a_i 为一个字节），输出 1 字的结果, 记为 $B = (b_0, b_1, b_2, b_3)$.

非线性变换就是对输入参数的每个字节进行 S 盒（Sbox）变换，得到输出结果，即 $B = (b_0, b_1, b_2, b_3) = \tau(A) = (Sbox(a_0), Sbox(a_1), Sbox(a_2), Sbox(a_3))$,S 盒如下图所示：

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	D6	90	E9	FE	CC	E1	3D	B7	16	B6	14	C2	28	FB	2C	05
1	2B	67	9A	76	2A	BE	04	C3	AA	44	13	26	49	86	06	99
2	9C	42	50	F4	91	EF	98	7A	33	54	0B	43	ED	CF	AC	62
3	E4	B3	1C	A9	C9	08	E8	95	80	DF	94	FA	75	8F	3F	A6
4	47	07	A7	FC	F3	73	17	BA	83	59	3C	19	E6	85	4F	A8
5	68	6B	81	B2	71	64	DA	8B	F8	EB	0F	4B	70	56	9D	35
6	1E	24	0E	5E	63	58	D1	A2	25	22	7C	3B	01	21	78	87
7	D4	00	46	57	9F	D3	27	52	4C	36	02	E7	A0	C4	C8	9E
8	EA	BF	8A	D2	40	C7	38	B5	A3	F7	F2	CE	F9	61	15	A1
9	E0	AE	5D	A4	9B	34	1A	55	AD	93	32	30	F5	8C	B1	E3

Figure 1: SM4 的 S 盒（第一部分）

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A	1D	F6	E2	2E	82	66	CA	60	C0	29	23	AB	0D	53	4E	6F
B	D5	DB	37	45	DE	FD	8E	2F	03	FF	6A	72	6D	6C	5B	51
C	8D	1B	AF	92	BB	DD	BC	7F	11	D9	5C	41	1F	10	5A	D8
D	0A	C1	31	88	A5	CD	7B	BD	2D	74	D0	12	B8	E5	B4	B0
E	89	69	97	4A	0C	96	77	7E	65	B9	F1	09	C5	6E	C6	84
F	18	F0	7D	EC	3A	DC	4D	20	79	EE	5F	3E	D7	CB	39	48

Figure 2: SM4 的 S 盒（第二部分）

具体代换方法与 AES 中的字节代替相同。首先把一个字节的 8 位二进制写成 2 位十六进制，然后以十六进制的第一个数字为行，第二个数字为列，在 S 盒表中查找对应的数字。这样就可以把一个字节代替成另一个字节。

(2) **线性变换** L

线性变换接收 1 字的 B 作为输入，经过运算，得出 1 字的输出 C.

运算公式如下：

$B = L(B) = B \oplus (B <<< 2) \oplus (B <<< 10) \oplus (B <<< 18) \oplus (B <<< 24)$

经过非线性变换和线性变换，就完成了一次轮迭代，计算出了下一个字的内容。

3.1.4 SM4 的密钥扩展算法

(1) **与系统参数异或——初始化密钥**

首先需要让原始密钥的每个字 MK_i 与系统参数 FK_i 异或，得到 4 个新的字 (K_0, K_1, K_2, K_3) ，即 $(K_0, K_1, K_2, K_3) = (MK_0 \oplus FK_0, MK_1 \oplus FK_1, MK_2 \oplus FK_2, MK_3 \oplus FK_3)$. 这里系统参数的取值为 $FK_0 = (A3B1BAC6), FK_1 = (56AA3350), FK_2 = (677D9197), FK_4 = (B27022DC)$

(2) **轮迭代生成轮密钥**

在对密钥进行初始化后，我们得到了 4 个新的字 (K_0, K_1, K_2, K_3), 之后，与加密过程类似，需要对这四个字进行 32 轮迭代，生成 32 个轮密钥。第一轮迭代为：

根据前面的 4 个字 (K_0, K_1, K_2, K_3) ，计算出第 5 个字 K_4 的值，并且将 K_4 作为第一轮的轮密钥 rk_0，计算方法如下：$rk_0 = K_4 = K_0 \oplus T'(K_1 \oplus K_2 \oplus K_3 \oplus CK_0)$.（这里的 T' 类似于合成置换 T,CK_0 为固定参数)

以此类推，一直进行 32 轮，直到得到 32 个轮密钥。这 32 个轮密钥就是密钥扩展算法的结果。其通式为：$rk_i = K_{i+4} \oplus T'(K_{i+1} \oplus K_{i+2} \oplus TK_{i+3} \oplus K_{i+4})$,

4 实验过程

4.1 编写 SM4 Verilog 代码

4.1.1 编写主体 SM4 mian 代码

首先定义输入输出,模块的输入信号包括时钟信号 CLK、复位信号 $RST\_N/SM4\_EN$ (使能 SM4 模块),128 位的明文数据输入 IN_DATA 和 128 位的加密密钥输入 IN_KEY。

模块的输出信号包括：128 位的密文数据输出 OUT_DATA 和表示输出数据正确性的信号 OUT_READY。

```
1  input   CLK,              // 时钟输入
2  input   RST_N,            // 复位信号输入
```

```
    input       SM4_EN,           // SM4加密使能信号输入
    input   [127:0] IN_DATA,       // 输入数据
    input   [127:0] IN_KEY,        // 输入密钥
    output  [127:0] OUT_DATA,      // 输出数据
```

模块中包含两个实例化的子模块：$F\_function\_rk$ 和 $F\_function$。具体来说，$F\_function\_rk$ 是一个扩展轮密钥模块，它接收来自顶层模块的时钟信号（CLK）、使能信号（Frk-EN）、输入数据（X0、X1、X2、X3）、密钥（CK）等，并输出中间寄存器的值（Y1、Y2、Y3、Y4）。$F\_function$ 是一个加密模块，它接收来自顶层模块的时钟信号（CLK）、使能信号（F-EN）、输入数据（X0、X1、X2、X3）、轮密钥（RK）等，并输出中间寄存器的值（Y1、Y2、Y3、Y4）。

在 main 中定义了一些内部信号（如 x1-l、x2-l 等）和控制信号（如 round-cnt、next-cnt 等），实现中间寄存器值的传递和状态机的控制。根据主循环（下文进行详细解释）的当前状态（round-cnt），控制 enc-en 信号的赋值和 reg-ck 的选择，从而实现对子模块 F-function-rk 和 F-function 的使能和控制。

主循环实现如下，round-cnt 寄存器用于表示当前的状态，通过赋值语句对其进行更新，控制状态的转移：

```
always @(posedge CLK or negedge RST_N) begin
    // 当复位信号 RST_N 为低电平时执行以下操作

        if (!RST_N) begin
            enc_en = 1'b0;// 加密使能信号置零
            next_cnt = 6'h00; // 下一个计数器值置零
            out_ready = 1'b0;// 输出准备信号置零
        end
        else if (SM4_EN == 1'b1) begin
            round_cnt = next_cnt;
            case (round_cnt)        // 根据当前轮数计数器的值进行不同的操作
                6'h00 : begin enc_en = 1'b0; next_cnt = 6'h01; out_ready = 1'b0; end
                6'h01 : begin enc_en = 1'b1; next_cnt = 6'h02; reg_ck = 32'h00070e15;end
                6'h02 : begin reg_ck = 32'h1c232a31;next_cnt = 6'h03; end
                6'h03 : begin reg_ck = 32'h383f464d;next_cnt = 6'h04; end
                6'h04 : begin reg_ck = 32'h545b6269;next_cnt = 6'h05; end
                6'h05 : begin reg_ck = 32'h70777e85;next_cnt = 6'h06; end
                6'h06 : begin reg_ck = 32'h8c939aa1;next_cnt = 6'h07; end
```

```
            6'h07 : begin reg_ck = 32'ha8afb6bd;next_cnt = 6'h08; end
            6'h08 : begin reg_ck = 32'hc4cbd2d9;next_cnt = 6'h09; end
            6'h09 : begin reg_ck = 32'he0e7eef5;next_cnt = 6'h0a; end
            6'h0a : begin reg_ck = 32'hfc030a11;next_cnt = 6'h0b; end
            6'h0b : begin reg_ck = 32'h181f262d;next_cnt = 6'h0c; end
            6'h0c : begin reg_ck = 32'h343b4249;next_cnt = 6'h0d; end
            6'h0d : begin reg_ck = 32'h50575e65;next_cnt = 6'h0e; end
            6'h0e : begin reg_ck = 32'h6c737a81;next_cnt = 6'h0f; end
            6'h0f : begin reg_ck = 32'h888f969d;next_cnt = 6'h10; end
            6'h10 : begin reg_ck = 32'ha4abb2b9;next_cnt = 6'h11; end
            6'h11 : begin reg_ck = 32'hc0c7ced5;next_cnt = 6'h12; end
            6'h12 : begin reg_ck = 32'hdce3eaf1;next_cnt = 6'h13; end
            6'h13 : begin reg_ck = 32'hf8ff060d;next_cnt = 6'h14; end
            6'h14 : begin reg_ck = 32'h141b2229;next_cnt = 6'h15; end
            6'h15 : begin reg_ck = 32'h30373e45;next_cnt = 6'h16; end
            6'h16 : begin reg_ck = 32'h4c535a61;next_cnt = 6'h17; end
            6'h17 : begin reg_ck = 32'h686f767d;next_cnt = 6'h18; end
            6'h18 : begin reg_ck = 32'h848b9299;next_cnt = 6'h19; end
            6'h19 : begin reg_ck = 32'ha0a7aeb5;next_cnt = 6'h1a; end
            6'h1a : begin reg_ck = 32'hbcc3cad1;next_cnt = 6'h1b; end
            6'h1b : begin reg_ck = 32'hd8dfe6ed;next_cnt = 6'h1c; end
            6'h1c : begin reg_ck = 32'hf4fb0209;next_cnt = 6'h1d; end
            6'h1d : begin reg_ck = 32'h10171e25;next_cnt = 6'h1e; end
            6'h1e : begin reg_ck = 32'h2c333a41;next_cnt = 6'h1f; end
            6'h1f : begin reg_ck = 32'h484f565d;next_cnt = 6'h20; end
            6'h20 : begin reg_ck = 32'h646b7279;next_cnt = 6'h21; end
            6'h21 : begin enc_en = 1'b0; out_ready = 1'b1; end
        endcase
    end
  end
```

主循环的初始进入的状态为 0，表示需要进行密钥扩展操作。在状态 0 下，控制信号 enc-en 被赋值为 0，使得子模块 F-function-rk 启动，进行轮密钥扩展操作。同时，输入信号 reg-ck 被选择为 CK，用于读取初始密钥。

在状态 0 下，会根据控制信号 Frk-EN 和 enc-en 的赋值将它们传入子模块 F-function-rk 中。在该子模块中，轮密钥扩展操作会执行若干次，每次都会产生 4 个

32 位输出，以及四个中间寄存器 Y1、Y2、Y3、Y4 的新值。在状态机中，更新控制信号和中间寄存器的值，等待下一轮操作。

当循环进入到状态 1，表示加密操作开始。在状态 1 下，控制信号 enc-en 被赋值为 1，使得子模块 F-function 被使能，进行轮加密操作。同时，输入信号 reg-ck 被选择为 Y4，用于读取前一轮加密的输出。

在状态 1 下，当输入数据 X0、X1、X2、X3 和轮密钥 RK 被准备好之后，会根据控制信号 F-EN 和 enc-en 的赋值将它们传入子模块 F-function 中。在该子模块中，轮加密操作会执行若干次，每次都会产生 4 个 32 位输出，以及四个中间寄存器 Y1、Y2、Y3、Y4 的新值。在循环中，更新控制信号和中间寄存器的值，等待下一轮操作。

4.1.2 编写 SM4 加密模块 (F 函数) 的 Verilog 代码

在本部分，实现 SM4 中 F 函数 (轮函数)，在主循环中，如果 F-EN 为 1，则进入 F 函数的执行流程，根据 3.1.2 的结构描述，轮函数 $F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i)$ 接收 5 个 1 字的参数，前 4 个字 $(X_i, X_{i+1}, X_{i+2}, X_{i+3})$ 为明文字或者迭代中间值，最后一个参数为 1 字的轮密钥 rk_i 。轮函数的输出结果为 1 字。轮函数内部需要执行的运算为 $F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i) = X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i)$, 公式后边的 T 为合成置换。

上述操作在代码中，则为：

1. 将输入数据 X1、X2 和 X3 与轮密钥 RK 进行异或操作，结果存储在 wire-xor 中。

2. 将 wire-xor 分别作为索引输入到四个 S 盒模块（t-sbox1、t-sbox2、t-sbox3、t-sbox4）中，从预定义的 S 盒中查找对应的输出值。各个 S 盒模块分别处理 wire-xor 的高 8 位，将结果存储在 wire-t-func 中，并执行 L 函数的置换和异或操作，最后于 X0 异或，得到 F 的输出

代码实现如下图所示：

```
module f(
    input          CLK,              // 时钟信号
    input          F_EN,              // F_EN使能信号

    input   [31:0] X0,               // 输入X0
    input   [31:0] X1,               // 输入X1
    input   [31:0] X2,               // 输入X2
    input   [31:0] X3,               // 输入X3
    input   [31:0] RK,               // 轮密钥RK
```

```
    output  [31:0]  Y1,             // 输出Y1
    output  [31:0]  Y2,             // 输出Y2
    output  [31:0]  Y3,             // 输出Y3
    output  [31:0]  Y4              // 输出Y4
);
    wire    [31:0]  wire_xor;           // 异或结果的中间信号
    wire    [31:0]  wire_t_func;        // T函数的中间信号
    wire    [31:0]  wire_L_func;        // L函数的中间信号
    wire    [31:0]  wire_result_Y4;     // Y4的中间信号

    S_BOX t_sbox1(CLK, wire_xor[31:24], wire_t_func[31:24]); // 实例化S_BOX模块用于计算
      t_sbox1
    S_BOX t_sbox2(CLK, wire_xor[23:16], wire_t_func[23:16]); // 实例化S_BOX模块用于计算
      t_sbox2
    S_BOX t_sbox3(CLK, wire_xor[15:8], wire_t_func[15:8]);  // 实例化S_BOX模块用于计算
      t_sbox3
    S_BOX t_sbox4(CLK, wire_xor[7:0], wire_t_func[7:0]);    // 实例化S_BOX模块用于计算
      t_sbox4

    assign wire_xor =  X1 ^ X2 ^ X3 ^ RK; // 计算异或结果
    assign wire_L_func = ( ( wire_t_func ^ {wire_t_func[29:0], wire_t_func[31:30]})
                          ^ ({wire_t_func[21:0], wire_t_func[31:22]} ^ {wire_t_func[13:0],
      wire_t_func[31:14]}))
                          ^ {wire_t_func[7:0], wire_t_func[31:8]}; // 计算L函数结果
    assign wire_result_Y4 = X0 ^ wire_L_func; // 计算Y4结果

    assign Y1 = (F_EN == 1'b1) ? X1 : X0; // 根据F_EN使能信号选择输出Y1
    assign Y2 = (F_EN == 1'b1) ? X2 : X1; // 根据F_EN使能信号选择输出Y2
    assign Y3 = (F_EN == 1'b1) ? X3 : X2; // 根据F_EN使能信号选择输出Y3
    assign Y4 = (F_EN == 1'b1) ? wire_result_Y4 : X3; // 根据F_EN使能信号选择输出Y4

endmodule
```

4.1.3 编写 SM4 密钥扩展模块的 Verilog 代码

生成轮密钥的密钥拓展函数写法与 F 函数大致相同，同样需要使用四个 S 盒模块 t-sbox1、t-sbox2、t-sbox3 和 t-sbox4 来分别处理 CK 的高 8 位、次高 8 位、次低 8 位和低 8 位。

将输入的 32 位密钥 CK 拆分为四个字 W0, W1, W2, W3。对 W1, W2, W3 进行迭代运算，生成轮密钥 W4 - W31。迭代运算的过程如下：

1. 对 W3 进行循环左移 9 位，并与 S 盒输出结果进行异或运算。

2. 将异或运算的结果与 W0 进行异或运算，得到新的轮密钥 W4。

3. 依次类推，对前面生成的轮密钥和 W0 进行异或运算，得到后续的轮密钥。最终生成 W4, W5, ..., W31。

```
assign wire_xor =  X1 ^ X2 ^ X3 ^ CK;  // 计算异或运算结果
assign wire_L_func = (wire_t_func ^ {wire_t_func[18:0], wire_t_func[31:19]}) ^ {wire_t_func
  [8:0], wire_t_func[31:9]};  // 计算L函数结果
assign wire_result_Y4 = X0 ^ wire_L_func;  // 计算Y4结果

assign Y1 = (Frk_EN == 1'b1) ? X1 : X0; // 根据使能信号选择输出数据Y1
assign Y2 = (Frk_EN == 1'b1) ? X2 : X1; // 根据使能信号选择输出数据Y2
assign Y3 = (Frk_EN == 1'b1) ? X3 : X2; // 根据使能信号选择输出数据Y3
assign Y4 = (Frk_EN == 1'b1) ? wire_result_Y4 : X3; // 根据使能信号选择输出数据Y4
```

4.1.4 编写 S 盒的 Verilog 代码

对于 S 盒，只需要利用 case 语句根据输入数据的不同值分别将对应的结果赋值给 result-reg 变量即可，如下：

```
always @( * )
   case(IN_DATA)
      8'h00 : result_reg = 8'hd6;  8'h01 : result_reg = 8'h90;  8'h02 : result_reg = 8'he9;  8'
  h03 : result_reg = 8'hfe;
      8'h04 : result_reg = 8'hcc;  8'h05 : result_reg = 8'he1;  8'h06 : result_reg = 8'h3d;  8'
  h07 : result_reg = 8'hb7;
      8'h08 : result_reg = 8'h16;  8'h09 : result_reg = 8'hb6;  8'h0a : result_reg = 8'h14;  8'
  h0b : result_reg = 8'hc2;
      8'h0c : result_reg = 8'h28;  8'h0d : result_reg = 8'hfb;  8'h0e : result_reg = 8'h2c;  8'
  h0f : result_reg = 8'h05;
```

```
7
8           ...
9           ...//(此处省略类似结构代码)
10
11          8'hf0 : result_reg = 8'h18;  8'hf1 : result_reg = 8'hf0;  8'hf2 : result_reg = 8'h7d;  8'
    hf3 : result_reg = 8'hec;
12          8'hf4 : result_reg = 8'h3a;  8'hf5 : result_reg = 8'hdc;  8'hf6 : result_reg = 8'h4d;  8'
    hf7 : result_reg = 8'h20;
13          8'hf8 : result_reg = 8'h79;  8'hf9 : result_reg = 8'hee;  8'hfa : result_reg = 8'h5f;  8'
    hfb : result_reg = 8'h3e;
14          8'hfc : result_reg = 8'hd7;  8'hfd : result_reg = 8'hcb;  8'hfe : result_reg = 8'h39;  8'
    hff : result_reg = 8'h48;
15        endcase
16
17 assign OUT_DATA = result_reg;
```

4.2 编写 SM4 仿真测试代码

接下来，编写对应的仿真测试代码：

```
1  `timescale 1ns/100ps
2  module top_tb;
3      reg             clk             ;
4      reg             reset_n         ;
5      reg             sm4_en          ;
6      reg   [127: 0]  data_in         ;
7      reg   [127: 0]  user_key_in     ;
8      wire            ready_out       ;
9      wire  [127: 0]  result_out      ;
10     //always #1  clk = ~clk;
11 initial begin
12     clk = 0;
13     #10 clk = ~clk;
14         ...//此处为压缩篇幅，省略部分时钟控制
15     #10 clk = ~clk;
16   end
17     main uut(.CLK(clk), .RST_N(reset_n), .SM4_EN(sm4_en), .IN_DATA(data_in), .IN_KEY(
```

```
    user_key_in), .OUT_DATA(result_out), .OUT_READY(ready_out));
  initial begin
    clk = 0;
    reset_n = 1'b0;
    sm4_en = 1'b0;
    #10
    sm4_en = 1'b1;
    reset_n = 1'b1;
    data_in = 128'h 01_23_45_67_89_ab_cd_ef_fe_dc_ba_98_76_54_32_10;
    user_key_in = 128'h 01_23_45_67_89_ab_cd_ef_fe_dc_ba_98_76_54_32_10;
    #10
    reset_n = 1'b0;
    sm4_en = 1'b0;
    #10
    reset_n = 1'b1;
    sm4_en = 1'b1;
    data_in = 128'h 01_23_45_67_89_ab_cd_ef_fe_dc_ba_98_76_54_32_10;
    user_key_in = 128'h 01_23_45_67_89_ab_cd_ef_fe_dc_ba_98_76_54_32_10;
    #10
    $stop;
  end
 initial begin
  $dumpfile("test.vcd");
  $dumpvars;
 end
endmodule
```

5 结果分析

5.1 SM4 仿真结果

运行该代码，在 iverlog 中显示密文为：

681edf34d206965e86b3e94f536e4246002a8a4efa863ccad024ac0300bb40d2：

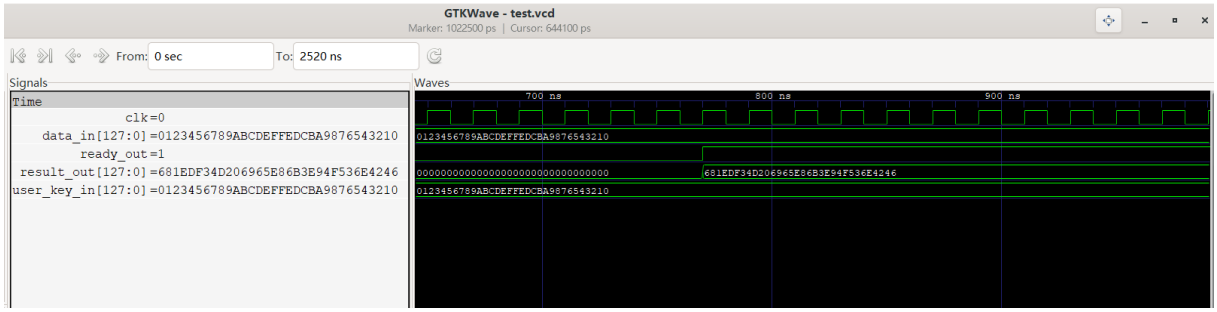

Figure 3: 仿真结果

与在线网站的 SM4 加密结果对比一致，说明代码加密成功

Figure 4: 在线加密网站

参考文献

[1] SM4 分组密码算法国家标准 GB/T 32907-2016

[2] Openssl 官方文档. https://www.openssl.org/docs/man1.1.1/man7/