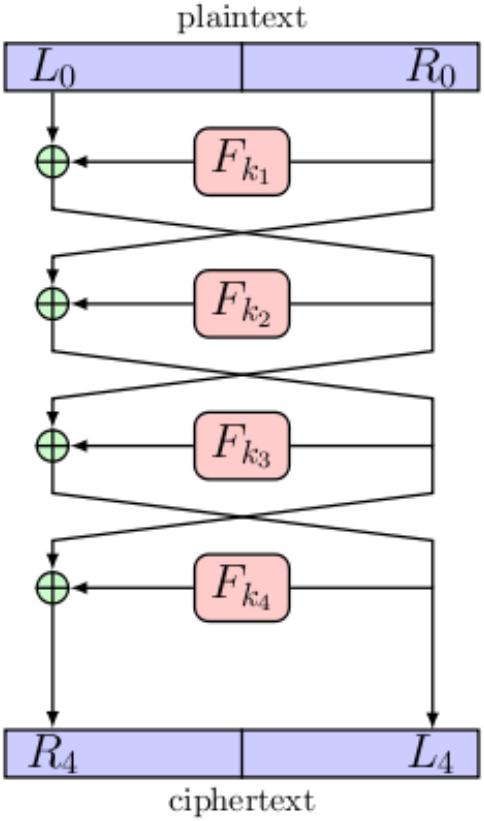
درس طراحی کامپیوتری سیستم‌های دیجیتال

گزارش شماره‌ی: 16

اعضای گروه: نرگس قاسمی ، تارا برقیان

چکیده: در این آزمایش، هدف پیاده سازی یک قطعه سخت افزاری بود که به حالت فیستل یک ورودی را با کلید مشخص رمز کرده و خروجی میدهد.

چالش اصلی پیاده سازی زیر کلید ها بود که در ادامه به توضیح آن میپردازیم.



فهرست مطالب

۱ - ورودی و خروجی ها

در این بخش علاوه بر سیگنال های تک بیتی کلاک، ریست، **start** یک کلید **128** بیتی و ورودی **64** بیتی داریم که قرار است رمز شود.

```
entity tea is

port(
     clk, reset : in std_logic;
     start : in std_logic;
     ready : out std_logic;
     key   : in std_logic_vector(127 downto 0);
     value  : in std_logic_vector(63 downto 0);
     out_ans : out std_logic_vector(63 downto 0)
);
end entity;
```

2- بدنه کد

کد از **2 process** اصلی تشکیل شده است.
پراسس اول وظیفه ی چک کردن تغییرات کلاک و ریست و چک کردن سیگنال **start** را دارد.
در صورت تغییر ریست و تغییر کلاک:
اگر لبه بالا رونده باشد و ریست **1** باشد مقدار شمارنده **31** میشود و در حالت استیت **start** قرار میگیرد.
اگر لبه بالا رونده باشد و **start 1** باشد، مقدار شمارنده **0** میشود و در حالت استیت **start** قرار میگرد و
در غیر این صورت اگر اگر لبه بالا رونده باشد مقدار**next_state** را قرار میدهد و مقدار شمارنده یکی زیاد میشود.

```
    process(clk, reset)
    begin

if rising_edge(clk) then
        if reset = '1' then
        state_reg <= start_state;
               count_reg <= (others => '1');

        elsif start = '1' then

              count_reg <= (others => '0');
           state_reg <= start_state;
        else
            state_reg <= state_next;
           count_reg <= count_next;
        end if;
     end if;


    end process;
```

در قسمت بعدی که **y** و **z** به هم وابسته هستند باید **variable** باشند که حالت ترتیبی داشته باشیم.(طبق قوانین **variable** ها)

```
process(count_reg, state_reg)
variable z,zt,y,yt: std_logic_vector (31 downto 0);
```

در این پراسس به تغییر شمارنده و استیت فعلی حساسیم. توجه شود که دقیق تر است اول کلاک زده شود و شمارنده اضافه شود، سپس تغییرات دیگر اعمال شوند که دقیقا همین روند اتفاق میفتد. (علاوه بر جواب درست شبیه سازی صحیح هم دارد.)

اگر سیگنال استارت فعال بود، مقدار دهی های اولیه یعنی تقسیم کلید به **4** بخش، و مقدار دادن دو طرف چپ و راست کلید به **z,y** صورت میگیرد. و سیگنال **ready** در ابتدا **0** است. (پس از پایان حلقه **1** میشود.)

```
if( state_reg = start_state )then
        sum  <= x"9e3779b9";
        count_next <= (others => '0');

        state_next <= count_state;

        k0 <= key(31  downto  0);
        k1 <= key(63  downto 32);
        k2 <= key(95  downto 64);
        k3 <= key(127 downto 96);
        y  := value(31  downto  0);
        z  := value(63  downto 32);
        ready <= '0';
```

اگر در حالت استارت نباشیم، یعنی در یکی از مرحله های رمز کردن هستیم، که عملیات طبق کد **c++** که داده شده بود در نظر گرفته شده است. دقت شود که تمامی سیگنال های **z,zt,y,yt** به صورت **variable** باید تعریف میشد.

```
else
    count_next <= count_reg  + 1;
    sum <= sum + x"9e3779b9";

        zt:= ((z(27 downto 0) &"0000")+k0) xor (z+sum) xor (("00000"& z(31 downto 5))+k1);
        y := y+zt;

        yt:= ((y(27 downto 0) &"0000")+k2) xor (y+sum) xor (("00000"& y(31 downto 5))+k3);
        z := z+yt;
```

در همین پراسس، تعداد حلقه های چرخیده شده چک میشوند تا از **32** کمتر باشد.

اگر **32** بار چرخیده بودیم فیسل کامل اجرا شده و جواب اماده است. همچنین مجدد قرار است به استیت استارت برگردیم پس، **next_state** با **start_state** مقدار دهی میشود.

```
        if (count_reg  + 1) = 32 then
            state_next <= start_state;
    ready <='1';
```

اگر هنوز **32** بار کامل نشده باشد، **ready** همچنان **0** خواهد بود و به شمارنده یکی اضافه میشود.

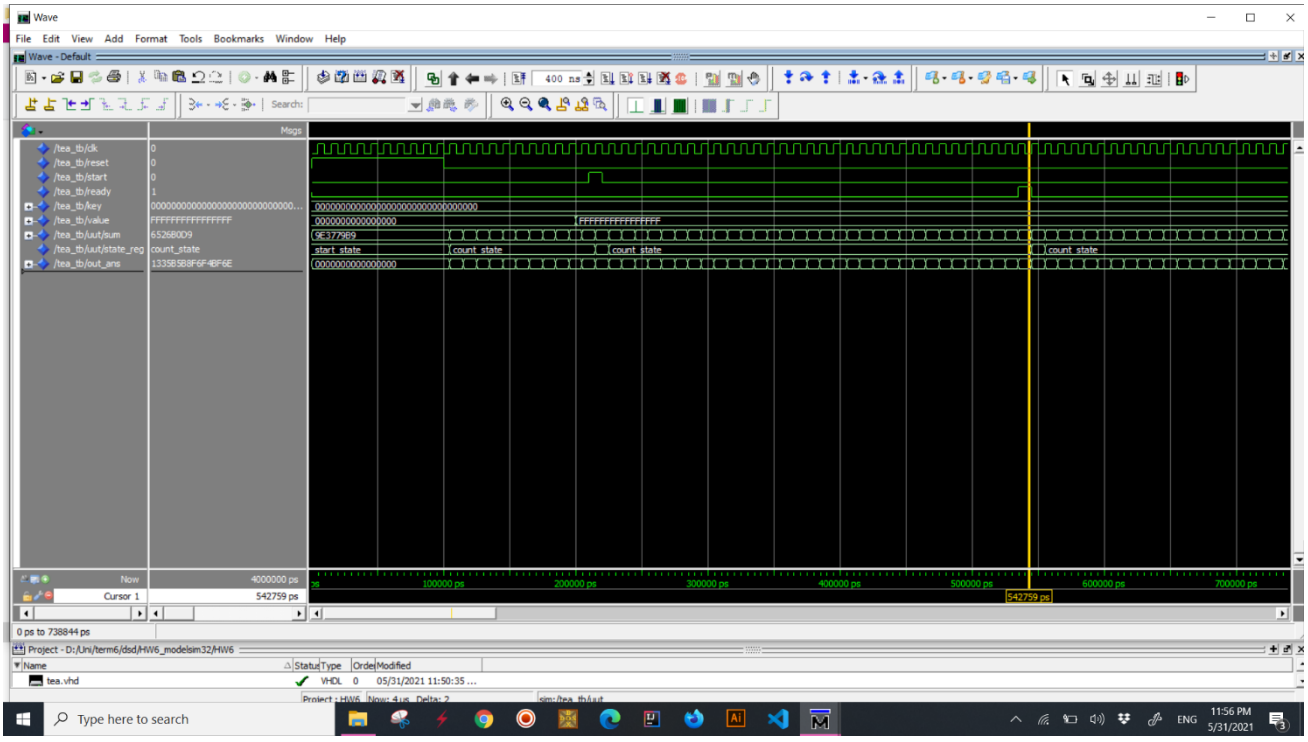
```
    else
        state_next <= count_state;
ready<='0';
```

در پایان فیسل، جواب از کانکت **z** و **y**حاصل میشود :

```
out_ans <= std_logic_vector(z&y);
```

3- تست و نتیجه گیری

ما **32** بار پس از **1** کردن استارت کلاک بالارونده اجرا کردیم و جواب به دست امده را با جواب کد **c++** مقایسه کردیم که یکسان بود :



z	y	sum
6428d3c1	663779b6	9e3779b9
5b13982f	47733773	3c6ef372
ce1337ae	cdce52de	daa66d2b
e1496f72	6f8150ad	78dde6e4
a6023fd8	5b43bd01	1715609d
762dfd05	99a0b351	b54cda56
29f601d8	427da1fd	5384540f
684d753a	c81c042b	f1bbcdc8
13114702	47f188dd	8ff34781
7c40e6f5	b8a27b01	2e2ac13a
e5b11f9c	47f1c490	cc623af3
e0d91d3c	54686a04	6a99b4ac
1c698b21	37a5ba8c	8d12e65
450f5761	3caf8802	a708a81e
53472131	1541fd95	454021d7
f37ccebf	1598946d	e3779b90
62c8b5cb	5b157dfa	81af1549
eda9e8c	347dbcd	1fe68f02
7f0c0ecd	296f7740	be1e08bb
2f25100f	51c89527	5c558274
10cc476a	2c62b973	fa8cfc2d
600a8947	d235633e	98c475e6
50a17440	66e41c1a	36fbef9f
7ff71bde	942aae54	d5336958
d3071615	a41aaa25	736ae311
b4e9c8da	765ad564	11a25cca
eb16799c	a653321f	afd9d683
bf2dd34c	356c15f3	4e11503c
20955457	91bedd0e	ec48c9f5
eb420bb	350354e5	8a8043ae
3e161023	115fc97c	28b7bd67
1335b5b8	f6f4bf6e	c6ef3720

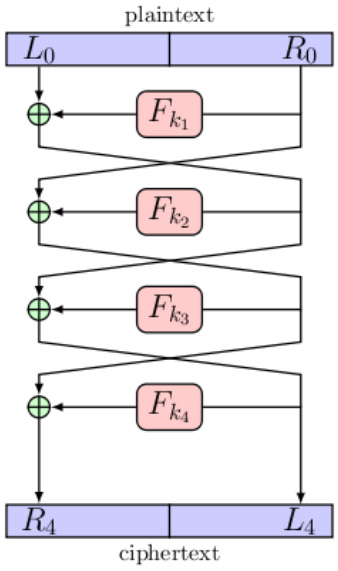
```c
#include<stdio.h>
#include <stdint.h>

uint32_t arr[]={0,0};
void code(uint32_t * v, uint32_t * k)
{
uint32_t  y=v[0] ;
uint32_t  z=v[1];
uint32_t  sum=0; /* set up */
uint32_t  delta=0x9e3779b9; /* a key schedule constant */
int n=32 ;
    while (n-->0)
    { /* basic cycle start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
    }

v[0]=y ; v[1]=z ;

arr[0]=y;
arr[1]=z;
}

int main(){
    uint32_t  v[]={0xFFFFFFFF,0xFFFFFFFF};
    uint32_t  k[]={0,0,0,0};
    code(v,k);
    printf("%x %x",arr[1],arr[0]);
    ///arr[1]= 1335b5b8
    ///arr[0]= f6f4bf6e
    return 0;
}
```




```
zt:= ((z(27 downto 0) &"0000")+k0) xor (z+sum) xor (("00000"& z(31 downto 5))+k1);
y := y+zt;

yt:= ((y(27 downto 0) &"0000")+k2) xor (y+sum) xor (("00000"& y(31 downto 5))+k3);
z := z+yt;
```

زیر کلید های ساخته شده در هر بخش از فیستل