مصطفی نجفی 98243089
پارسا نوری 98243067

در این تمرین اقدام به پیاده‌سازی RSA با استفاده از هم‌طراحی سخت افزار و نرم‌افزار نموده‌ایم. انجام اعمال gcd و power را به سخت‌افزار سپرده‌ایم. برای این منظور پورت‌های مربوط به ورودی و خروجی این دو را در فایل c و در ابتدای فایل fdl تعریف می‌کنیم.

در فایل c:

```c
// gcd memory map
volatile unsigned long long int *gcd_in_a = (unsigned long long int *) 0x80000000;
volatile unsigned long long int *gcd_in_b = (unsigned long long int *) 0x80000008;
volatile unsigned long long int *gcd_out = (unsigned long long int *) 0x80000010;
volatile unsigned  int *gcd_load = (unsigned int *) 0x80000018;
volatile unsigned  int *gcd_done = (unsigned int *) 0x8000001c;

// power memory map
volatile unsigned long long int *pow_in_b = (unsigned long long int *) 0x80000020;
volatile unsigned long long int *pow_in_e = (unsigned long long int *) 0x80000028;
volatile unsigned long long int *pow_in_m = (unsigned long long int *) 0x80000030;
volatile unsigned long long int *pow_out = (unsigned long long int *) 0x80000038;
volatile unsigned  int *pow_load = (unsigned int *) 0x80000040;
volatile unsigned  int *pow_done = (unsigned int *) 0x80000044;
```

در فایل .fdl :

```
ipblock myarm{
    iptype "armsystem";
    ipparm "exec=rsa";
    ipparm "period=1";
}

ipblock arm_gcd_in_a(out data : ns(64)){
    iptype "armsystemsource";
    ipparm "core=myarm";
    ipparm "address=0x80000000";
}

ipblock arm_gcd_in_b(out data : ns(64)){
    iptype "armsystemsource";
    ipparm "core=myarm";
    ipparm "address=0x80000008";
}

ipblock arm_gcd_out(in data : ns(64)){
    iptype "armsystemsink";
    ipparm "core=myarm";
    ipparm "address=0x80000010";
}

ipblock arm_gcd_load(out data : ns(1)){
    iptype "armsystemsource";
    ipparm "core=myarm";
    ipparm "address=0x80000018";
}

ipblock arm_gcd_done(in data : ns(1)){
    iptype "armsystemsink";
    ipparm "core=myarm";
    ipparm "address=0x8000001c";
}
```

```
ipblock arm_pow_in_b(out data : ns(64)){
    iptype "armsystemsource";
    ipparm "core=myarm";
    ipparm "address=0x80000020";
}

ipblock arm_pow_in_e(out data : ns(64)){
    iptype "armsystemsource";
    ipparm "core=myarm";
    ipparm "address=0x80000028";
}

ipblock arm_pow_in_m(out data : ns(64)){
    iptype "armsystemsource";
    ipparm "core=myarm";
    ipparm "address=0x80000030";
}

ipblock arm_pow_out(in data : ns(64)){
    iptype "armsystemsink";
    ipparm "core=myarm";
    ipparm "address=0x80000038";
}

ipblock arm_pow_load(out data : ns(1)){
    iptype "armsystemsource";
    ipparm "core=myarm";
    ipparm "address=0x80000040";
}

ipblock arm_pow_done(in data : ns(1)){
    iptype "armsystemsink";
    ipparm "core=myarm";
    ipparm "address=0x80000044";
}
```

در فایل c آدرس مربوط به هر پورت به یک پوینتر assign شد و در فایل fdl به صورت یک سری ipblock که یک سری به صورت ورودی و یک سری دیگر به صورت خروجی هستند.

در main دو عدد اول رندوم انتخاب شده تا سپس با کمک آن‌ها d و e انتخاب شوند.

```
// Two random prime numbers
unsigned long long int p = 5;
unsigned long long int q = 7;
// First part of public key
```

طبق الگوریتم این دو در هم ضرب شده و n نامیده می‌شود و ک.م.م یکی کم‌تر از این دو z نامیده شده است.

```
// First part of public key
unsigned long long int n = p*q;
// Finding other part of public key
unsigned int long long e = 2;
unsigned long long int z = lcm(p-1,q-1); // results in smaller d than
just multiplying them ▸a: p-1 ▸h: q-1
```

اولین عددی که نسبت به z اول باشد به عنوان e انتخاب می‌شود.

```
while (e < z) {
    // e must be co-prime to z and smaller than z
    if (gcd(e, z)==1) ▸a: e ▸b: z
        break;
    else
        e++;
}
```

اولین عددی که اگر در e ضرب شود باقی‌مانده‌اش در تقسیم با z برابر یک شود به عنوان d انتخاب می‌شود.

```
//point 2

unsigned long long int d = 2;
while (d < z) {
    if ((d*e)%z == 1)
        break;
    else
        d++;
}
```

پیام تعریف شده.

```
// Message to be encrypted
unsigned long long msg = (long long int)'!';
printf("Message data = %lld\n", msg);

// point 4
```

پیام رمزنگاری شده برابر باقی‌مانده تقسیم msg به توان d بر n می‌باشد.

```
// Encryption c = (msg ^ e) % n
unsigned long long c  = power(msg, e, n); ▸b: msg ▸m: n
// c %= n;
printf("Encrypted data = %lld\n", c);
```

پیام رمزگشایی شده نیز معادل باقی مانده تقسیم پیام رمزنگاری شده به توان e و بر n می‌باشد.

```
// point 5

// Decryption m = (c ^ d) % n
unsigned long long m = power(c, d, n); ▸b: c ▸e: d ▸m: n
// m %= n;
printf("Original Message Sent = %lld\n", m);

return 0;
```

منتها مسئله اصلی در اینجاست که توابع gcd و power چگونه کار می‌کنند.

برای gcd داریم:

```
76 unsigned long long int gcd(unsigned long long int a, unsigned long long int
   b){
77     *gcd_in_a = a;
78     *gcd_in_b = b;
79     *gcd_load = 1;
80     while (*gcd_done !=1 );
81     long long int result = *gcd_out;
82     *gcd_load = 0;
83     // printf("[debug] gcd(%lld, %lld) = %lld\n", a, b, result);
84     return result;
85 }
```

این تابع در پورت‌های module مربوط به gcd مقادیری که بایستی میانگینشان گرفته شده را می‌گذارد و سپس load آن را فعال می‌کند. منتظر اتمام کار gcd می‌ماند و سپس با استفاده از پورت gcd_out نتیجه را می‌خوانند و سپس load مربوط به gcd را فعال می‌کند.

برای power داریم:

```
91 unsigned long long power(unsigned long long b, unsigned long long e, unsigned
   long long m){
92     *pow_in_b = b;
93     *pow_in_e = e;
94     *pow_in_m = m;
95     *pow_load = 1;
96     while (*pow_done !=1 );
97     long long int result = *pow_out;
98     *pow_load = 0;
99     // printf("[debug] pow(%lld, %lld, %lld) = %lld\n", b, e, m, result);
00     return result;
01 }
```

مقادیر b و e و m که به ترتیب base و power و modulus می‌باشند را روی پورت‌های مربوطه گذاشته و منتظر load آن را فعال می‌نماید سپس منتظر می‌ماند که کار انجام شود و سپس حاصل را می‌خواند. سپس مقدار load را برای استفاده بعدی 0 می‌نماید.

در فایل fdl ماژول‌های مربوطه پیاده‌اسزی شده‌اند. یعنی gcd و power در آن‌ها پیاده سازی شده‌اند.

دری gcd داریم:

```
fsm gcd_fsm(gcd_dp) {
    initial s0;
    state s1, s2;
    @s0
        if (load) then (init) -> s1;
        else (ready) -> s0;
    @s1
        if (a%b==0) then (outp_idle) -> s2;
        else (loop, log) -> s1;
    @s2 if (load) then (outp_idle) -> s2;
        else (ready) -> s0;
}
```

در s0 منتظر یک شدن load برای شروع فرایند می‌ماند. سپس با تنظیم a و b به عنوان متغیرهای درونی اقدام به رفتن به s1 می‌کند. در s1 در صورتی که باقیمانده تقسیم a بر b صفر باشد، به s2 رفته و فرایند تمام است و در غیر این صورت در عمل loop هر بار باقی‌مانده a بر b در b ریخته شده و b به داخل a ریخته می‌شود. در s2 تا وقتی که ready صفر نشده سیگنال آماده نشان داده می‌شود و در غیر این صورت به وضعیت s0 برگشت داده می‌شود.

در dp مربوط به gcd اعمال گفته شده پیاده سازی شده است.

```
77 dp gcd_dp(
78     in  a_in   :   ns(64);
79     in  b_in   :   ns(64);
80     in  load:   ns(1);
81     out o   :   ns(64);
82     out done:   ns(1)
83 ) {
84     reg a, b: ns(64);
85 
86     always {
87         o = b;
88     }
89 
90     sfg ready {
91         done = 0;
92     }
93 
94     sfg init {
95         a = a_in;
96         b = b_in;
97         done = 0;
98     }
99 
100    sfg loop {
101        a = b;
102        b = a%b;
103        done = 0;
104    }
105
106    sfg outp_idle {
107        done = 1;
108    }
109
110    sfg log {
111        $display("GCD: ", "cycle=", $cycle, " a=", a, " b=", b, " load=",
           load, " o=", o, " done=", done);
112    }
113 }
```

در fsm مربوط به power مشروط به اینکه load فعال باشد فرایند شروع شده و i=0 , acc=۱ و done=0 می‌شود. در غیر این صورت در حالت آماده باقی می‌ماند. سپس در s۱ تا وقتی که i==e نشده هر بار یکی به i اضافه شده و acc یک بار در base ضرب می شود در غیر این صورت done=1 شده و به s2 وارد شده و در آن‌جا تا وقتی که load صفر نشده done=1 قرار داده می‌شود تا اینکه load صفر شود و به s0 وارد شود.

```
fsm pow_fsm(pow_dp) {
    initial s0;
    state s1, s2;

    @s0
        if (load) then (init) -> s1;
        else (ready) -> s0;
    @s1
        if (i==e) then (outp_idle) -> s2;
        else (loop,log) -> s1;
    @s2 if (load) then (outp_idle) -> s2;
        else (ready) -> s0;
}
```

در dp مربوط به pow اعمال گفته شده پیاده سازی شده است.

```
dp pow_dp (
    in  b    :   ns(64);
    in  e    :   ns(64);
    in  m    :   ns(64);
    in  load:   ns(1);
    out o    :   ns(64);
    out done:   ns(1)
) {
    reg acc, i : ns(64);

    always {
        o = acc;
    }

    sfg ready {
        done = 0;
    }

    sfg init {
        acc = 1;
        i = 0;
        done = 0;
    }

    sfg loop {
        acc = (acc * b) % m;
        i = i + 1;
        done = 0;
    }

    sfg outp_idle {
        done = 1;
    }

    sfg log {
        $display("POWER: ","cycle=", $cycle, " b=", b, " e=", e, " m=", m, "
        load=", load, " o=", o, " acc=", acc, " done=", done);
    }
```

کل تمرین با استفاده از run.sh اجرا می‌شود. خروجی به شکل زیر است:

```
--> compiling sw
--> run gplatform
GCD: cycle=6469 a=4/6 b=6/4 load=1 o=6 done=0
GCD: cycle=6470 a=6/4 b=4/2 load=1 o=4 done=0
GCD: cycle=6810 a=2/c b=c/2 load=1 o=c done=0
GCD: cycle=6867 a=3/c b=c/3 load=1 o=c done=0
GCD: cycle=6923 a=4/c b=c/4 load=1 o=c done=0
GCD: cycle=6979 a=5/c b=c/5 load=1 o=c done=0
GCD: cycle=6980 a=c/5 b=5/2 load=1 o=5 done=0
GCD: cycle=6981 a=5/2 b=2/1 load=1 o=2 done=0
Message data = 33
POWER: cycle=11162 b=21 e=5 m=23 load=1 o=1 acc=1/21 done=0
POWER: cycle=11163 b=21 e=5 m=23 load=1 o=21 acc=21/4 done=0
POWER: cycle=11164 b=21 e=5 m=23 load=1 o=4 acc=4/1b done=0
POWER: cycle=11165 b=21 e=5 m=23 load=1 o=1b acc=1b/10 done=0
POWER: cycle=11166 b=21 e=5 m=23 load=1 o=10 acc=10/3 done=0
Encrypted data = 3
POWER: cycle=12772 b=3 e=5 m=23 load=1 o=1 acc=1/3 done=0
POWER: cycle=12773 b=3 e=5 m=23 load=1 o=3 acc=3/9 done=0
POWER: cycle=12774 b=3 e=5 m=23 load=1 o=9 acc=9/1b done=0
POWER: cycle=12775 b=3 e=5 m=23 load=1 o=1b acc=1b/b done=0
POWER: cycle=12776 b=3 e=5 m=23 load=1 o=b acc=b/21 done=0
Original Message Sent = 33
--> done
root@5a18d379c0f4:/home/rsa-codesign#
```