



تمرین سوم هم طراحی سخت افزار - نرم افزار

نیلوفر مرادی جم 97243063

کیمیا صدیقی 97243046

مقدمه

در این تمرین، الگوریتم RSA که الگوریتمی برای رمزنگاری و رمزگشایی یک پیام است، به صورت هم سخت افزاری و هم نرم افزاری پیاده سازی شده است. به دلیل سریعتر بودن سخت افزار در انجام محاسبات، محاسبات اصلی به صورت سخت افزاری پیاده سازی می شوند.

کد سخت افزاری

ابتدا ipblock های مورد نیاز که برای تعریف و هم گام سازی با پردازنده ARM است تعریف می شوند. Ipblock ها هر کدام برای ما حکم راه ارتباطی بین سخت افزار و نرم افزار را دارند یعنی به آدرسی مشابه آدرسی که در نرم افزار گفته شده اشاره میکنند و بلاک ها را پر می کنند. (در نرم افزار ما از این طریق مقادیر را میگیریم)

```
1 ipblock my_arm {
2     iptype "armsystem";
3     ipparm "exec=rsadrive";
4 }
5
6 ipblock arm_gcd_out(in data: ns(32)) {
7     iptype "armsystemsink";
8     ipparm "core=my_arm";
9     ipparm "address=0x80000000";
10 }
11
12 > ipblock arm_gcd_strb(in data: ns(1)) {-
16 }
17
18 > ipblock arm_power_out(in data: ns(32)) {-
22 }
23
24 > ipblock arm_power_strb(in data: ns(1)) {-
28 }
29
30 > ipblock arm_point1_out(in data: ns(32)) {-
34 }
35
36 > ipblock arm_point1_strb(in data: ns(1)) {-
40 }
41
42 > ipblock arm_point2_out(in data: ns(32)) {-
46 }
47
48 > ipblock arm_point2_strb(in data: ns(1)) {-
52 }
53
54 > ipblock arm_e_out(out data: ns(32)) {-
58 }
```

در ادامه datapath های مورد نیاز تعریف می شوند. برای هر قسمت یک datapath مجزا تعریف کردیم. (برای datapath ها یک خروجی با پسوند strb نیز تعریف کردیم که هر زمان وظایف آن datapath تمام میشود، این خروجی مقدار یک میگیرد.)

اولین datapath مربوط به محاسبه GCD است. این تابع دو عدد 32 بیتی به عنوان ورودی می گیرد و بزرگترین مقسوم علیه مشترک آن دو را به عنوان خروجی برمیگرداند. متغیر init که در تمام دیتاپت های دیگر نیز مورد استفاده قرار گرفته، برای کنترل حلقه میباشد، که بار اول که صفر است مقدار a_register را برابر a و مقدار h_register را برابر h گذاشتیم و در دفعات بعدی که وارد این بلاک always میشویم این مقادیر تغییر میکنند یعنی از آن بعد a_register مقدار h قبلی را که در h_register بوده نگه میدارد و h_register مقدار جدید a/h را میگیرد. (این مربوط به الگوریتم محاسبه مقسوم علیه میباشد).

```

72 //      this datapath computes gcd of the numbers
73 dp gcd(in a: ns(32);
74       in h: ns(32);
75       out d_output: ns(32);
76       out d_output_strb: ns(1)) {
77
78     sig temp: ns(32);
79     reg a_register, h_register, init: ns(32);
80
81     always {
82       a_register = (init == 0) ? a : h_register;
83       h_register = (init == 0) ? (h == 0) ? 1 : h : temp;
84
85       temp = a_register % h_register;
86       d_output = (temp == 0) ? h_register : d_output;
87
88       init = 1;
89     }
90 }

```

datapath بعدی مربوط به محاسبه توان است که دو عدد را ورودی گرفته و اولی را به توان دومی رسانده و نتیجه را به عنوان خروجی برمیگرداند.

```

91
92 //      this datapath computes power of the numbers
93 dp power(in a: ns(32);
94         in n: ns(32);
95         out d_output: ns(32);
96         out d_output_strb: ns(1)) {
97
98     reg power, i, d_output_register, init: ns(32);
99
100    always {
101      power = (init == 0) ? a : power * a;
102      i = (init == 0) ? 1 : i + 1;
103
104      d_output_strb = (i < n) ? 0 : 1;
105      d_output_register = (d_output_strb) ? power : d_output_register;
106
107      d_output = d_output_register;
108
109      init = 1;
110    }
111
112 }
113

```

قسمت اصلی الگوریتم RSA نیز در دو datapath مجزا هندل می شود، یکی مربوط به point1 و دیگری مربوط به point2.

در point1 مقدار e محاسبه می شود.

```
114 //      this datapath calculates e according to point one of main program
115 dp point1(in e: ns(32);
116          in z: ns(32);
117          out d_output: ns(32);
118          out d_output_strb: ns(1)) {
119
120     sig gcd_out: ns(32);
121     sig gcd_strb: ns(1);
122     reg e_register, init: ns(32);
123
124     use gcd(e, z, gcd_out, gcd_strb);
125
126     always {
127         e_register = (init == 0) ? e : (gcd_out == 1) ? e_register : e_register + 1;
128         d_output = e_register;
129         d_output_strb = (e_register >= z) ? 1 : (gcd_out == 1) ? 1 : 0;
130
131         init = 1;
132     }
133 }
134
135
```

در point2 مقدار d را محاسبه می کنیم.

```
136 //      this datapath calculates d according to point one of main program
137 dp point2(in d: ns(32);
138          in z: ns(32);
139          in e: ns(32);
140          out d_output: ns(32);
141          out d_output_strb: ns(1)) {
142
143     reg d_register, e_register, init, z_temp: ns(32);
144     sig temp: ns(32);
145
146     always {
147         d_register = (init == 0) ? d : d_register + 1;
148         e_register = (init == 0) ? e : e_register;
149
150         z_temp = (z == 0) ? 1 : z;
151
152         temp = (d_register * e) % z_temp;
153         d_output_strb = (e_register >= z) ? 1 : (temp == 1) ? 1 : 0;
154         d_output = (d_output_strb) ? d_register : 0;
155
156         init = 1;
157     }
158 }
159
160
```

در نهایت یک datapath نهایی داریم که تمامی این قسمت ها را متصل می کند.

کد نرم افزاری

در این قسمت، تعامل با کاربر و مقداردهی اولیه متغیرها از جمله p و q انجام می شود. ابتدا آدرس هایی از حافظه را که در سخت افزار نیز به صورت ipblock تعریف کردیم برای تعامل با سخت افزار و گرفتن مقادیر، تعریف میکنیم.

```
2
3 volatile unsigned int *gcd_out      = (unsigned int *) 0x80000000;
4 volatile unsigned int *gcd_strb     = (unsigned int *) 0x80000004;
5 volatile unsigned int *pow_out      = (unsigned int *) 0x80000008;
6 volatile unsigned int *pow_strb     = (unsigned int *) 0x8000000C;
7 volatile unsigned int *point1_out   = (unsigned int *) 0x80000010;
8 volatile unsigned int *point1_strb  = (unsigned int *) 0x80000014;
9 volatile unsigned int *point2_out   = (unsigned int *) 0x80000018;
10 volatile unsigned long int *point2_strb = (unsigned int *) 0x8000001C;
11 volatile unsigned long int *e_out   = (unsigned int *) 0x80000020;
12 volatile unsigned long int *z_out   = (unsigned int *) 0x80000024;
13 volatile unsigned long int *d_out   = (unsigned int *) 0x80000028;
14
```

طبق صورت پروژه الگوریتم rsa شامل 5 پوینت است که هرکدام را جدا در کد نرم افزار مدیریت کردیم.

در پوینت یک، مقدار e در سخت افزار محاسبه می شود و در آدرس مناسب در حافظه قرار می گیرد.

```
24
25 /* * * * * * * * * * * point 1 * * * * * */
26 *e_out = e;
27 *z_out = z;
28 while (!*point1_strb) {}
29
30 // final e
31 e = *point1_out;
32
```

در پوینت دو با توجه به مقدار محاسبه شده e در پوینت یک، مقدار d در سخت افزار محاسبه شده و در آدرس مناسب خود در حافظه قرار میگیرد.

```
33
34 /* * * * * * * * * * * point 2 * * * * * */
35 *d_out = 2;
36 *e_out = e;
37 *z_out = z;
38 while (!*point2_strb) {}
39 d = *point2_out;
40
41
```

در پوینت سه تنها متغیر msg مقداردهی می شود.

```
40
41
42 /* * * * * * * * * * * point 3 * * * * * */
43 unsigned int msg = 20;
44 printf("Message data = %d", msg);
45
```

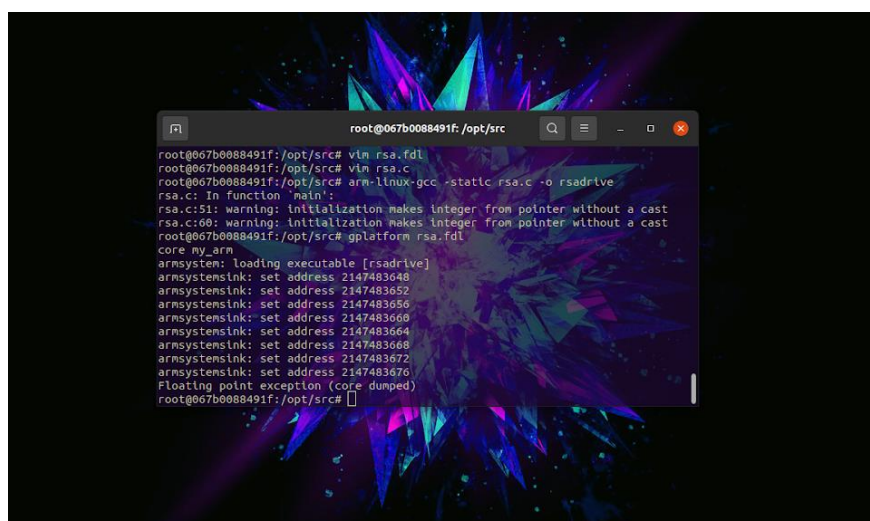
در پوینت چهار این مقدار رمزنگاری شده و در متغیر c قرار میگیرد.

```
46
47 /* * * * * * * * * * * point 4 * * * * * */
48 *e_out = msg;
49 *z_out = e;
50 while (!*pow_strb);
51 unsigned long long c = pow_out;
52 c = c % n;
53 printf("\n Encrypted data = %ld", c);
54
```

در پوینت پنج، مقدار رمزنگاری شده، رمزگشایی شده و در متغیر m قرار داده می شود.

```
55
56 /* * * * * * * * * * * point 5 * * * * * */
57 *e_out = c;
58 *z_out = d;
59 while (!*pow_strb);
60 unsigned long long m = pow_out;
61 m = m % n;
62 printf("\nOriginal message sent = %ld", m);
63
64
65 return 0;
66
```

خروجی:



```
root@067b0088491f: /opt/src
root@067b0088491f: /opt/src# vln rsa.fdl
root@067b0088491f: /opt/src# vln rsa.c
root@067b0088491f: /opt/src# arm-linux-gcc -static rsa.c -o rsadrive
rsa.c: In function 'main':
rsa.c:51: warning: initialization makes integer from pointer without a cast
rsa.c:60: warning: initialization makes integer from pointer without a cast
root@067b0088491f: /opt/src# gplatform rsa.fdl
core my_arm
armsystem: Loading executable [rsadrive]
armsystemsink: set address 2147483648
armsystemsink: set address 2147483652
armsystemsink: set address 2147483656
armsystemsink: set address 2147483660
armsystemsink: set address 2147483664
armsystemsink: set address 2147483668
armsystemsink: set address 2147483672
armsystemsink: set address 2147483676
Floating point exception (core dumped)
root@067b0088491f: /opt/src#
```

نتیجه گیری

ما در این پروژه یک سیستم کامل (به همراه نرم-افزار و سخت-افزار) را پیاده‌سازی کردیم. مزیت این روش نسبت به روش کاملاً نرم-افزار این است که محاسبات همه به وسیله‌ی سخت-افزار انجام میشوند و این باعث میشود که سرعت محاسبات بسیار بیشتر شود. همچنین مزیت این روش نسبت به روش کاملاً سخت-افزار این است که ورودی گرفتن و خروجی دادن بسیار ساده‌تر است و نیازی به درگیر شدن با LCD ها یا ابزار دیگر نیست.