گزارشِ تکلیفِ ششمِ هم‌طرّاحی

روزبه صیّادی - امیرعلی منجر

۱ مقدمه

هدف از این پروژه طرّاحیِ یک نرم‌افزار و یک سخت‌افزار است، به گونه‌ای که این دو همراه با هم کار کنند و نتیجه‌ی نهایی را به کمکِ هم به دست بیاورند. در چنین سیستمی معمولاً کارِ محاسبات به قسمتِ سخت‌افزار سپرده می‌شود که سریع‌تر است، و کارهای مقداردهیِ اوّلیّه و تعامل با کاربر در قسمتِ نرم‌افزار پیاده‌سازی می‌شود.
برای رسیدن به این منظور، الگوریتمِ رمزنگاریِ RSA را پیاده‌سازی کرده‌ایم.

۲ توضیحِ کد

۲,۱ نرم‌افزار

قسمتِ نرم‌افزار به زبانِ C نوشته شده است. در ابتدای کد آدرس‌های مربوط به پین‌های req و ack مقداردهی شده‌اند.

```
volatile unsigned int *req = (unsigned int *) 0x80000000;
volatile unsigned int *ack = (unsigned int *) 0x80000004;
```

دو تابع داریم که به ترتیب req را یک و صفر می‌کنند، و منتظرِ مقدارِ صفر و یک برای ack می‌مانند.

```
void sync1() {
   *req = 1;
   while ( *ack == 0 )
          ;
}

void sync0() {
   *req = 0;
   while ( *ack == 1 )
          ;
}
```

در تابعِ main ابتدا دو متغیّرِ دیگر برای پین‌های ورودی و خروجیِ پردازنده تعریف می‌کنیم.

```
volatile unsigned int *di = (unsigned int *) 0x80000008;
volatile unsigned int *ot = (unsigned int *) 0x8000000C;
```

سپس متغیّرهای p و q را طبقِ روشی که در کلاس آموختیم به پردازنده می‌دهیم.

```
*di = 13;
sync1();
*di = 17;
sync0();
```

سپس مقدار پیام را مشخّص می‌کنیم، منتظرِ محاسباتِ سخت‌افزار می‌مانیم و در انتها مقدارِ رمزشده‌ی پیام را چاپ می‌کنیم.

```
*di = 10;
sync1();
sync0();
printf("C: %ld\n", *ot);
```

در انتهای برنامه نیز منتظر می‌مانیم تا سخت‌افزار مقدارِ رمزگشایی شده‌ی عددِ به دست آمده (که قاعدتاً باید با پیامِ اصلی برابر باشد) را محاسبه کند. آن را در نرم‌افزار می‌گیریم و مقدارِ آن را چاپ می‌کنیم.

```
sync1();
printf("M: %ld\n", *ot);
sync0();
```

۲,۲ سخت‌افزار

قسمتِ سخت‌افزار کمی پیچیده‌تر از کدِ نرم‌افزار است.

۱.۲.۲ ARM Interface

تکه‌ی ابتداییِ کد برای تعریف و هم‌گام‌سازی با پردازنده‌ی ARM است.

```
ipblock my_arm {
   iptype "armsystem";
   ipparm "exec=rsadrive";
}

ipblock m_req(out data : ns(32)) {
   iptype "armsystemsource";
   ipparm "core=my_arm";
   ipparm "address=0x80000000";
}

ipblock m_ack(in data : ns(32)) {
   iptype "armsystemsink";
   ipparm "core=my_arm";
   ipparm "address=0x80000004";
}

ipblock m_data_out(out data : ns(32)) {
   iptype "armsystemsource";
   ipparm "core=my_arm";
   ipparm "address=0x80000008";
}

ipblock m_data_in(in data : ns(32)) {
```

```
  iptype "armsystemsink";
  ipparm "core=my_arm";
  ipparm "address=0x8000000C";
}
```

۲,۲,۲ داده‌مسیرِ rsa

حالا برای الگوریتمِ RSA یک داده‌مسیر[۱] تعریف می‌کنیم. این داده‌مسیر (rsa) وظیفه‌ی انجامِ محاسبات را دارد. عملکردِ بیشترِ ورودی‌ها و خروجی‌های آن قابلِ تشخیص هستند، ولی شاید چند تا از آن‌ها نیاز به توضیح داشته باشند:

- go_next:هر موقع یک شود، داده‌مسیر از مرحله‌ی محاسبه‌ی c واردِ مرحله‌ی محاسبه‌ی m می‌شود.
- init:مشخّص می‌کند که موقعِ مقداردهیِ اوّلیّه به متغیّرها هست یا نه.
- d_ready:مشخّص می‌کند که آیا مقدارِ d آماده هست یا نه.

در قسمتِ بعدی تعداد رجیستر تعریف شده است که از اسمشان مشخّص است چه نقشی دارند.

کدِ اصلیِ این داده‌مسیر در بلوکِ always قرار دارد.

- phase:اگر ورودی go_next فعّال شده باشد مقدارش یک می‌شود. در غیر این صورت همان مقدار قبلی را نگه می‌دارد.
- did_init:اگر ورودی init فعال شده باشد مقدارش یک می‌شود. در غیر این صورت همان مقدار قبلی را نگه می‌دارد.
- n:اگر مقدار init یک بود برابر با p*q می‌شود.
- z:اگر مقدار init یک بود مقدار (p-1)*(q-1)را ذخیره می‌کند.
- e:در حالت init مقدار (p-1)*(q-1)-1 را در خود ذخیره می‌کند (که هم نسبت به z کوچک‌تر باشد و هم نسبت به آن اوّل باشد). در غیر این صورت اگر قبلاً مقداردهی شده باشد یکی از مقدارش کم می‌شود تا به ۱- برسد. این کار برای محاسبه‌ی توانی که بعداً وجود دارد مفید است.
- d: اگر در حالت init باشد مقدار ۲ را می‌گیرد. در غیر این صورت اگر در فاز رمزنگاری باشیم تا وقتی که شرایط مورد نیاز برای الگوریتم را برآورده نکرده است یکی زیاد می‌شود. اگر در فاز رمزگشایی بودیم، از مقدارش یکی کم می‌شود (برای محاسبه‌ی توان).
- c: در حالت رمزنگاری مقدارش ۱ است. در غیر این صورت اگر قبلاً مقداردهی انجام گرفته بود آن را به توان می‌رسانیم و باقی‌مانده‌اش به n را در انتها حساب می‌کنیم. در غیر این صورت به مقدارش دست نمی‌زنیم.
- m: در فاز اوّلیّه مقدارش را صفر می‌گذاریم. در فاز دوم اگر مقداردهی اوّلیّه انجام گرفته شده بود آن را طوری تغییر می‌دهیم که شرایطِ الگوریتم را برآورده کند. در غیر این صورت به مقدار آن دست نمی‌زنیم.
- ready:اگر در فاز رمزنگاری مقدار e برابر با ۱- بود یعنی کارِ توان‌رسانی به پایان رسیده است و جواب آماده است. اگر در فاز رمزگشایی بودیم نیز اگر مقدار d صفر بود یعنی کار به اتمام رسیده است.
- d_ready: اگر مقداردهی‌های اوّلیّه انجام گرفته بودند و مقدار d شرایط الگوریتم را برآورده می‌کرد یعنی آماده است.
- result: بسته به این که در چه فازی قرار داریم مقدار c یا m را در خروجی قرار می‌دهیم.

---

[۱] Data path

۲,۲,۳ داده‌مسیرِ tb_rsa

این داده‌مسیر تعدادی sfg دارد که وظیفه و نحوه‌ی عملکرد هر کدام از روی اسمشان مشخّص است. وظیفه‌ی دیگرِ این داده‌مسیر اتّصالِ پردازنده به سیگنال‌های خود و همچنین تعریفِ یک instance از rsa و وصل کردنِ سیم‌های مناسب به آن است.

۲,۲,۴ ماشین حالت متناهی ctl_tb_rsa

در این قسمت ماشین حالتِ مورد نیاز برای الگوریتم توصیف می‌شود.
عملکرد این ماشین حالت به این صورت است که ابتدا دو عدد اوّل را از نرم‌افزار می‌گیرد. سپس این دو عدد را به داده‌مسیرِ قبلی می‌دهد تا متغیّرهایش را مقداردهی کند. سپس واردِ حالتِ compute_d می‌شود که باید منتظر بماند تا d محاسبه شود. بعد از انجامِ این محاسبه واردِ حالتِ گرفتنِ پیام می‌شود. حالتِ بعدی compute_c است که برای رمزگذاری روی پیام است.
بعد از محاسبه‌ی پیامِ رمزشده این مقدار باید به نرم‌افزار تحویل داده شود که در آن‌جا چاپ شود. این کار در حالتِ out_c طبقِ روشِ شرح داده شده در کلاس انجام می‌شود.
در انتها نیز در دو حالتِ باقی‌مانده پیامِ اصلی بازیابی می‌شود و روی صفحه نشان داده می‌شود. بعد از آن ماشین حالت به حالتِ ابتداییش بازمی‌گردد و منتظرِ ورودی‌های جدید می‌ماند.

۳ نتیجه‌گیری

ما در این پروژه یک سیستمِ کامل (به همراهِ نرم‌افزار و سخت‌افزار) را پیاده‌سازی کردیم. مزیت این روش نسبت به روشِ کاملاً-نرم‌افزار این است که محاسبات همه به وسیله‌ی سخت‌افزار انجام می‌شوند و این باعث می‌شود که سرعتِ محاسبات بسیار بیشتر شود. همچنین مزیت این روش نسبت به روشِ کاملاً-سخت‌افزار این است که ورودی گرفتن و خروجی دادن بسیار ساده‌تر است و نیازی به درگیر شدن با LCDها یا ابزارِ دیگر نیست.