# IDDMM算法模型设计

模乘运算：，在数论、群论、环论、代数、密码学、计算机科学等学科中都有着广泛地应用，从奇偶数的判别到素数的判别，从模幂运算到最大公约数的求法，都充斥这模运算的身影。

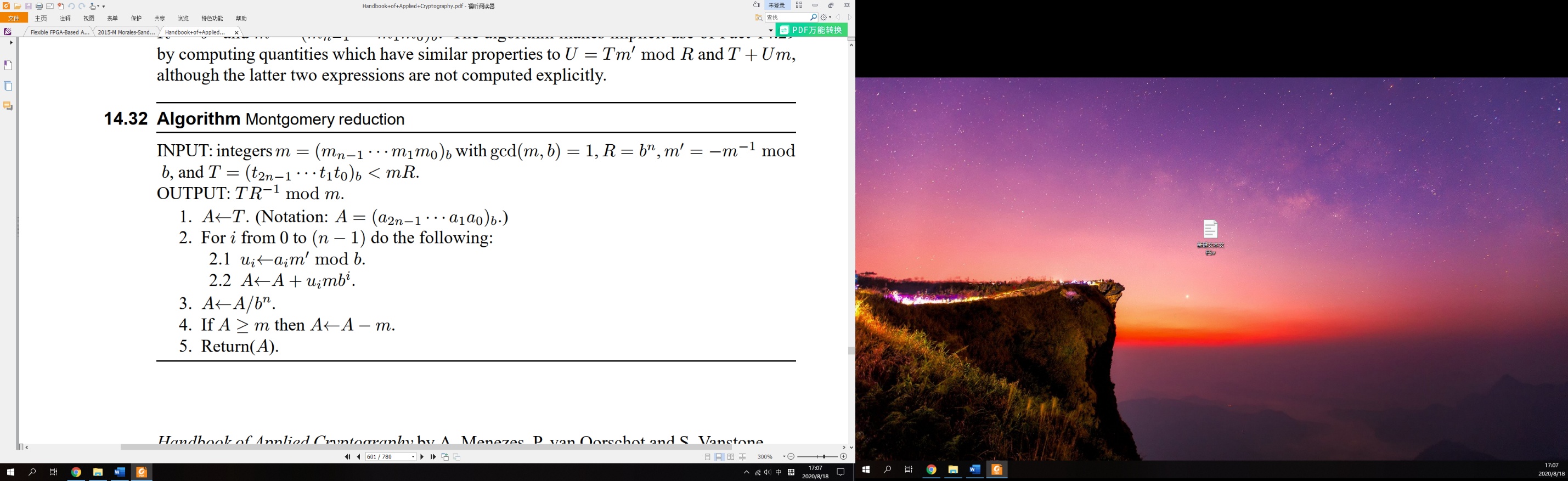
在蒙哥马利约减出现前，经典模乘算法被广泛应用。经典模乘算法使用大数计算方法，首先使用大数乘法，随后使用大数除法取余得到的结果。蒙哥马利约减是一种在不显式地执行经典模乘步骤的情况下实现模乘的一种技术，用于快速幂模运算，例如计算。蒙哥马利约减算法如下图1所示。

图1 蒙哥马利约简

假设和是在范围内的整数。让以及。的蒙哥马利约减结果是： 。考虑到计算，其中。首先计算然后计算的蒙哥马利约减，结果是：。的蒙哥马利约减结果是：最后(的蒙哥马利约减结果是,将它乘以然后模，就能得到的结果。在该算中，是一个进制长度为的整数，T也使用进制表示，的典型取值为，且。是求逆元后取负数模的结果。模值并不能取到任意值，在RSA算法中为奇数。

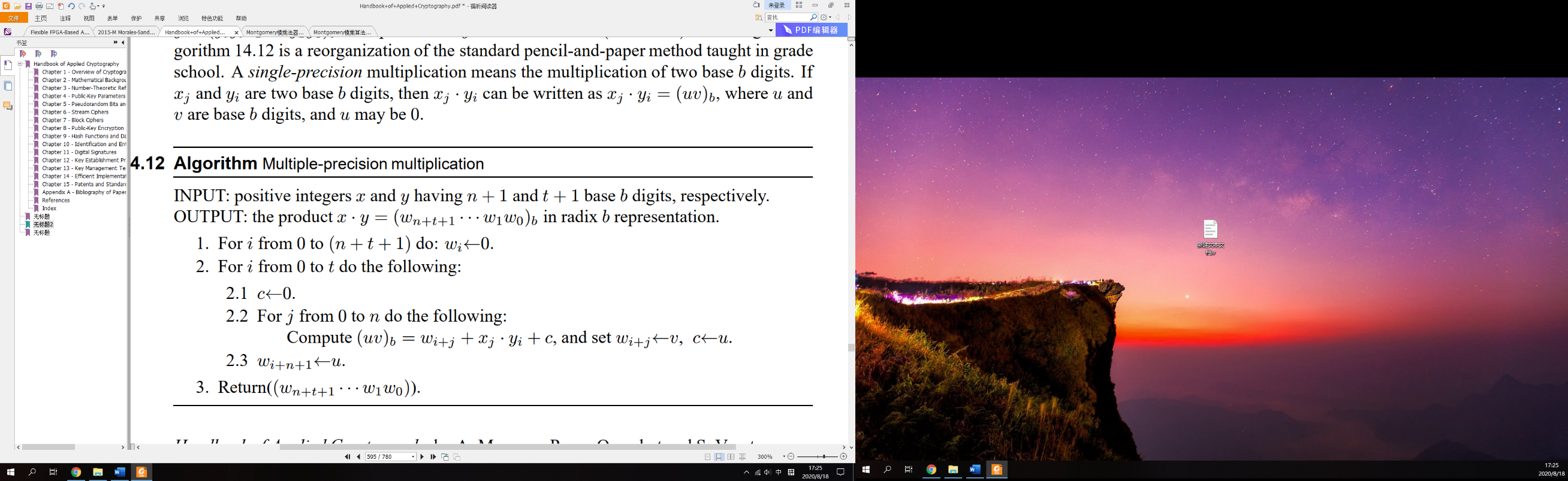
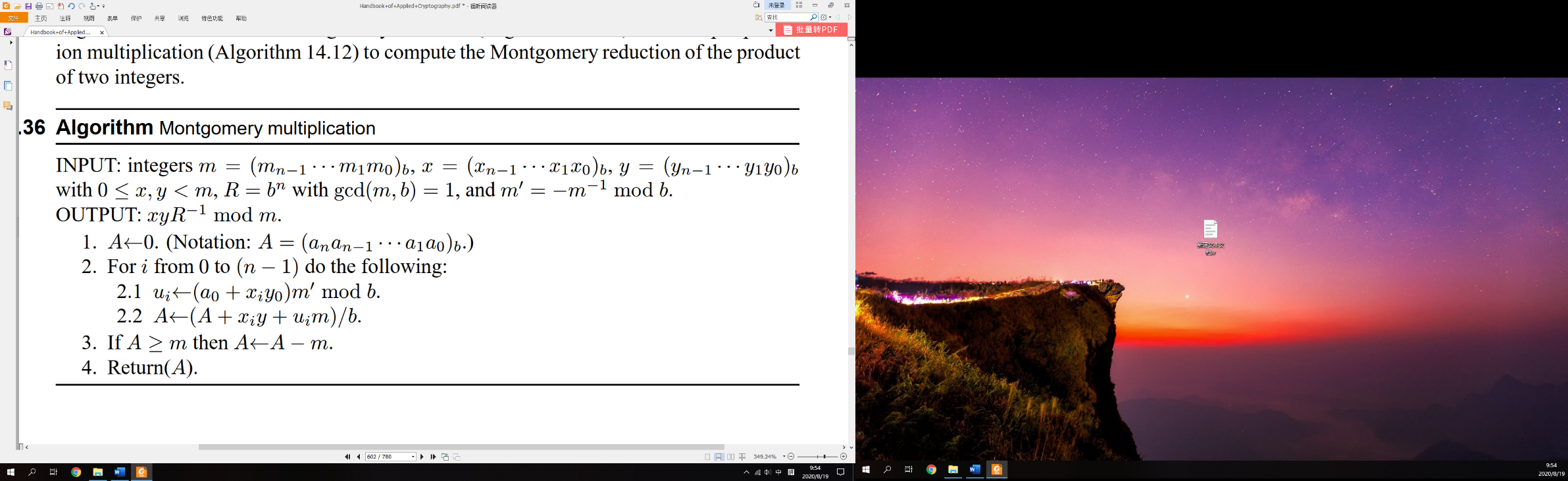
将替换为两个整数乘积后，可以使用手工乘法（中学乘法）图2，将蒙哥马利约减改写为蒙哥马利模乘，图3。

图2 手工乘法（中学乘法）

图3 Alfred J. Menezes蒙哥马利模乘

在图3蒙哥马利模乘中，是进制数的倍数，因此除以的操作可以由右移操作代替。在每次轮迭代中，和都会被计算一次，可以将展开为以下几步：

在步骤2中，结果需要模，是所有操作数的进制数，因此可以只取的最低位用于取模运算，由于小于进制数，因此模操作等效于取最低位。

在步骤3、4、5、6中，可以使用一个循环和中学乘法分别把每一位计算出来，可以发现该循环需要遍历次，加上外部遍历次，算法至少需要个周期。计算过程中，比特位数不会超过进制数所限定的范围。在本设计中，需要计算4096比特蒙哥马利模乘，且周期小于4000，可以取进制数，每位含有128比特，分组数，所有操作皆以128比特运算为基础。

在步骤7中，除法由右移完成。

2015年，M. Morales-Sandoval根据以上思想，提出了Iterative digit-digit Montgomery Multiplication(IDDMM)算法。2016年，Dorian Amiet纠正了IDDMM算法中进位链错误并改用了以上含有减法的运算。本设计采用了Dorian Amiet修正的IDDMM算法。

修正后的IDDMM算法如图4所示。在输入数据中，，。图4中的风别是模数和进制数。是对原数据的分组数。是的模逆元取负数后，再模的结果，该值可以预先计算，它是一个比特数。在IDDMM算法中是个位数，的最高分组位为0。该算法返回蒙哥马利模乘结果：。

在第9行中，位宽为比特，由于模值是进制数，模的结果就是后取低位，因此可以取的低位与相乘。同样技巧应用在第15、16行，这使得该算法十分适合硬件计算。通过改变分组数，IDDMM拥有灵活的缩放特性，可以设置不同的运算位宽以适应不同的硬件配置。较小的运算周期依赖于较大的运算位宽，同时会造成时序收敛困难。使用较小的运算位宽，易于时序收敛，但是运算周期会成倍增加。选取一个合适的分组数十分重要。在本设计中，输入位宽，分组数，单组比特数。算法核心部分需要经过1056次迭代。

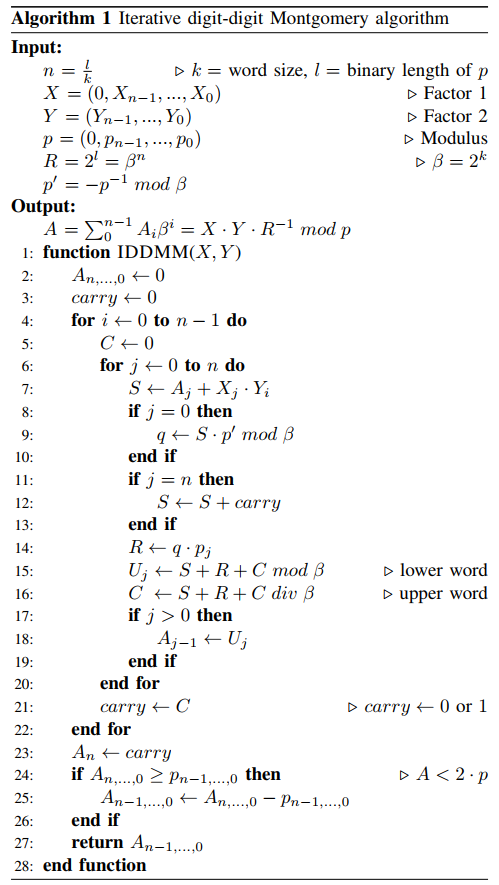


图4 本设计采用的蒙哥马利域模乘算法

为快速验证IDDMM算法的正确性，加速硬件验证，本设计使用Python程序语言对IDDMM算法进行建模。Python程序语言支持大数运算，可以限定运算位宽，符合硬件运算规律。

等效IDDMM算法Python等效代码：

1. **def** mont\_check(xx,yy,r,m,k):
2. **if** xx<m **and** yy<m **and** math.gcd(r,m)==1 **and** m>=2\*\*(k-1) **and** m<2\*\*k **and** r==2\*\*k:
3. **return** True
4. **else**:
5. **print**('x<m y<m gcd(r,m) m>=2^(k-1) m<2^k r==2^k')
6. **print**(xx<m,yy<m,math.gcd(r,m)==1, m>=2\*\*(k-1), m<2\*\*k,r==2\*\*k)
7. **return** False
8. **def** mod\_inv(a,m):
9. **if** math.gcd(a,m)!=1:
10. **return** None
11. u1,u2,u3 = 1,0,a
12. v1,v2,v3 = 0,1,m
13. **while** v3!=0:
14. q = u3//v3
15. v1,v2,v3,u1,u2,u3 = (u1-q\*v1),(u2-q\*v2),(u3-q\*v3),v1,v2,v3
16. **return** u1%m
17. **def** mm(x,k):
18. '''''
19. 对x取低k位
20. '''
21. **return** x&(2\*\*k-1)
22. **def** mont\_iddmm(xx,yy,p,nbit,n):
23. '''''
24. 这是Dorian Amiet改写的iddmm算法,修正了carry bit的问题
25. 文章：Flexible FPGA-Based Architectures for Curve Point Multiplication over GF(p)
26. return xx\*yy\*2^(-nbit) mod p
27. n      是分组数
28. '''
29. k=nbit//n
30. beta=2\*\*k
31. p1=(-1\*(mod\_inv(p,beta)))%beta
32. a =[0 **for** i **in** range(n+1)]
33. carry=0
34. x\_=[0 **for** i **in** range(n+1)]
35. y\_=[0 **for** i **in** range(n  )]
36. p\_=[0 **for** i **in** range(n+1)]
37. **for** i **in** range(n):
38. x\_[i]=mm(xx>>(i\*k),k)
39. y\_[i]=mm(yy>>(i\*k),k)
40. p\_[i]=mm(p >>(i\*k),k)
41. **for** i **in** range(n):
42. c=0
43. **for** j **in** range(n+1):
44. s=a[j]+x\_[j]\*y\_[i]
45. s=mm(s,2\*k)
46. **if** j==0:
47. q=mm(s,k)\*mm(p1,k)%beta
48. **if** j==n:
49. s=s+carry
50. r   = q\*p\_[j]
51. buf0= s+r+c
52. u   =  mm(buf0,k)
53. c   = (buf0>>k)
54. **if** j>0:
55. a[j-1]=u
56. carry=c&1
57. a[n]=carry
58. res=a
59. su =0
60. **for** i **in** range(len(res)):
61. su=su+res[i]\*(beta\*\*i)
62. **if** su >= p:
63. su=su-p
64. **return** su

以上Python代码截取自文件：\Montgomery\src\montgomery\_mul\_hd.py。

函数mont\_iddmm(xx,yy,p,nbit,n)即蒙哥马利域模乘算法IDDMM。函数形参xx、yy皆为蒙哥马利域数字；p是模数；nbit是总位宽，在本设计中固定4096比特；n是分组数，在本设计中固定32组。

函数mm(x,k)用于截取位宽。函数mod\_inv(a,m)用于对a求模m的逆元。函数mont\_check(xx,yy,r,m,k)用于判断输入数据是否符合运算前提，代码中收紧了模数m的范围用来适应基2蒙哥马利算法，在IDDMM算法中，输入数据只要满足上文提到的范围即可。

运行montgomery\_mul\_hd.py。代码将会比较是否与直接计算 的结果相等。Python代码中还包含了其他蒙哥马利域模乘算法的对比，用于参考。

为了解决现实问题：，需要在蒙哥马利域进行4次运算：