# IDDMM-MMP硬件设计文档

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。

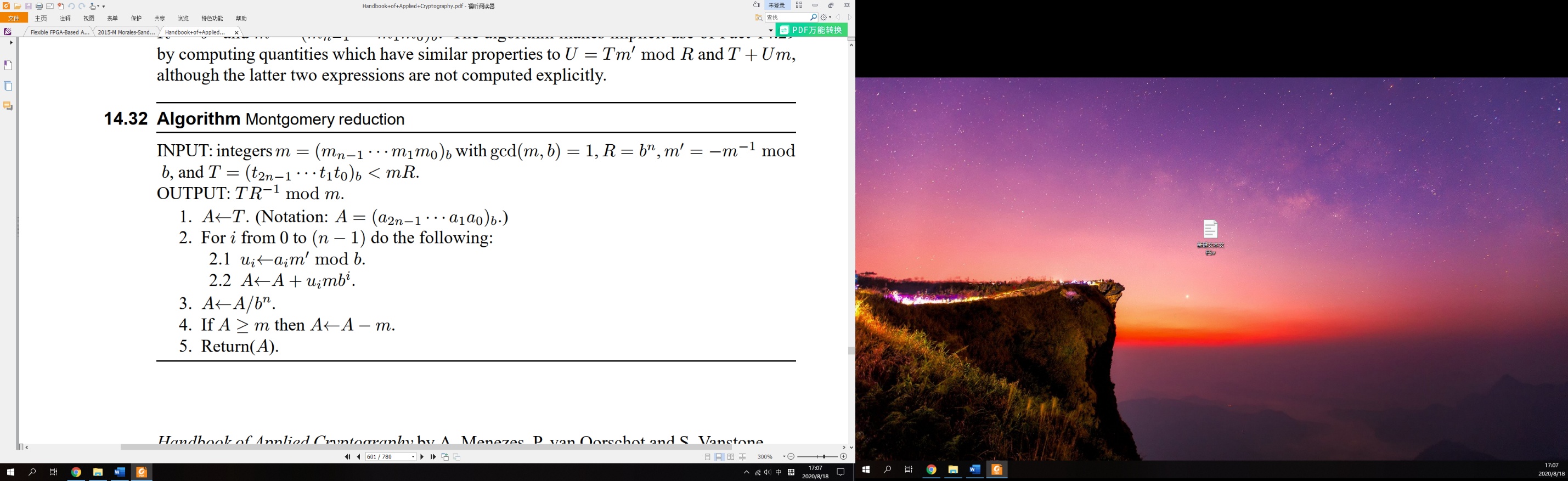
在蒙哥马利约减出现前，经典模乘算法被广泛应用。经典模乘算法使用大数计算方法，首先使用大数乘法，随后使用大数除法取余得到的结果。蒙哥马利约减是一种在不显式地执行经典模乘步骤的情况下实现模乘的一种技术，用于快速幂模运算，例如计算。蒙哥马利约减算法如下图1所示。

图1 蒙哥马利约简

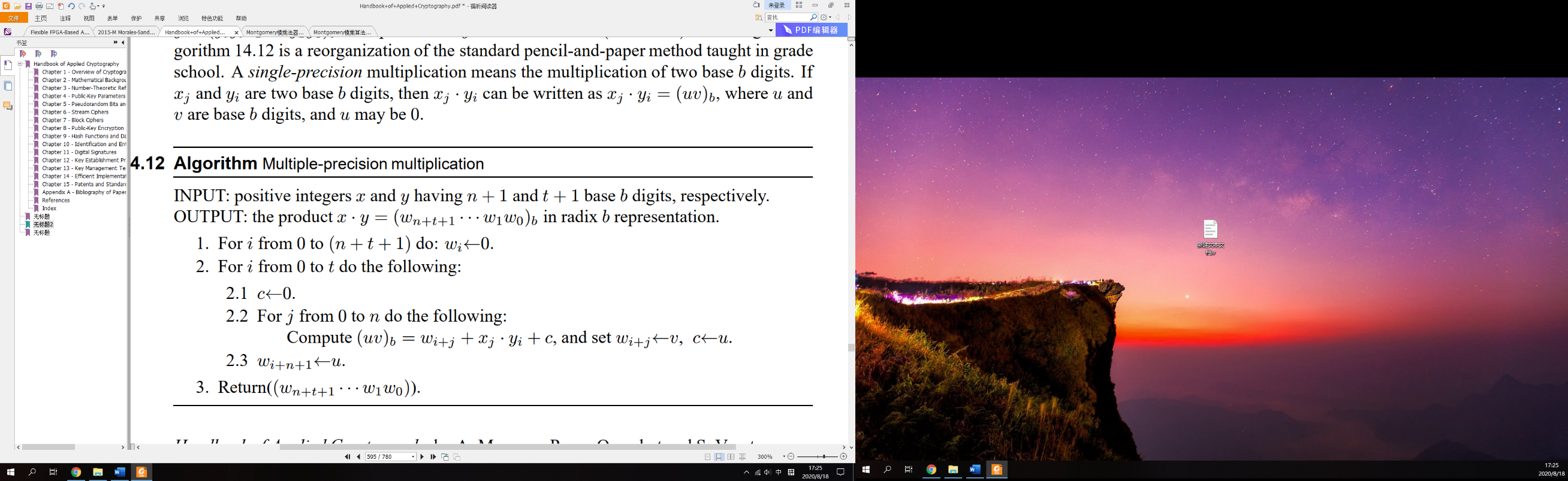
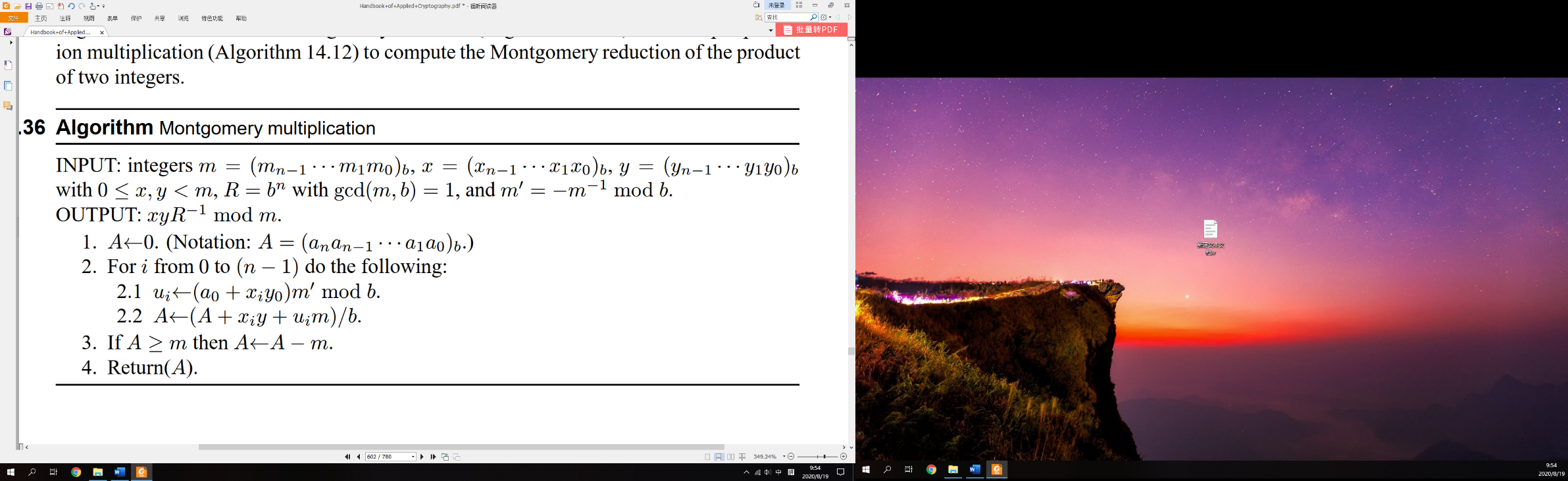
将替换为两个整数乘积后，可以使用手工乘法（中学乘法）图2，将蒙哥马利约减改写为蒙哥马利模乘，图3。

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。

本设计IDDMM算法使用了流水线即Montgomery multiplication pipeline，将该实现命名为IDDMM-MMP。

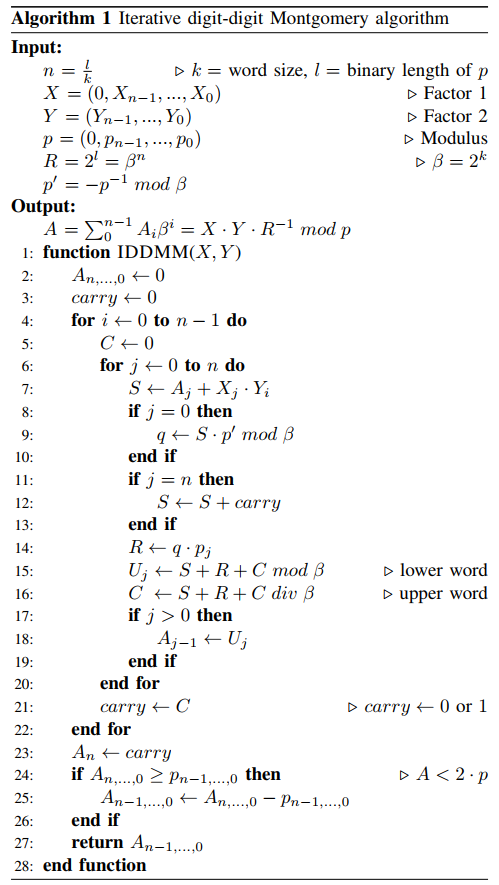


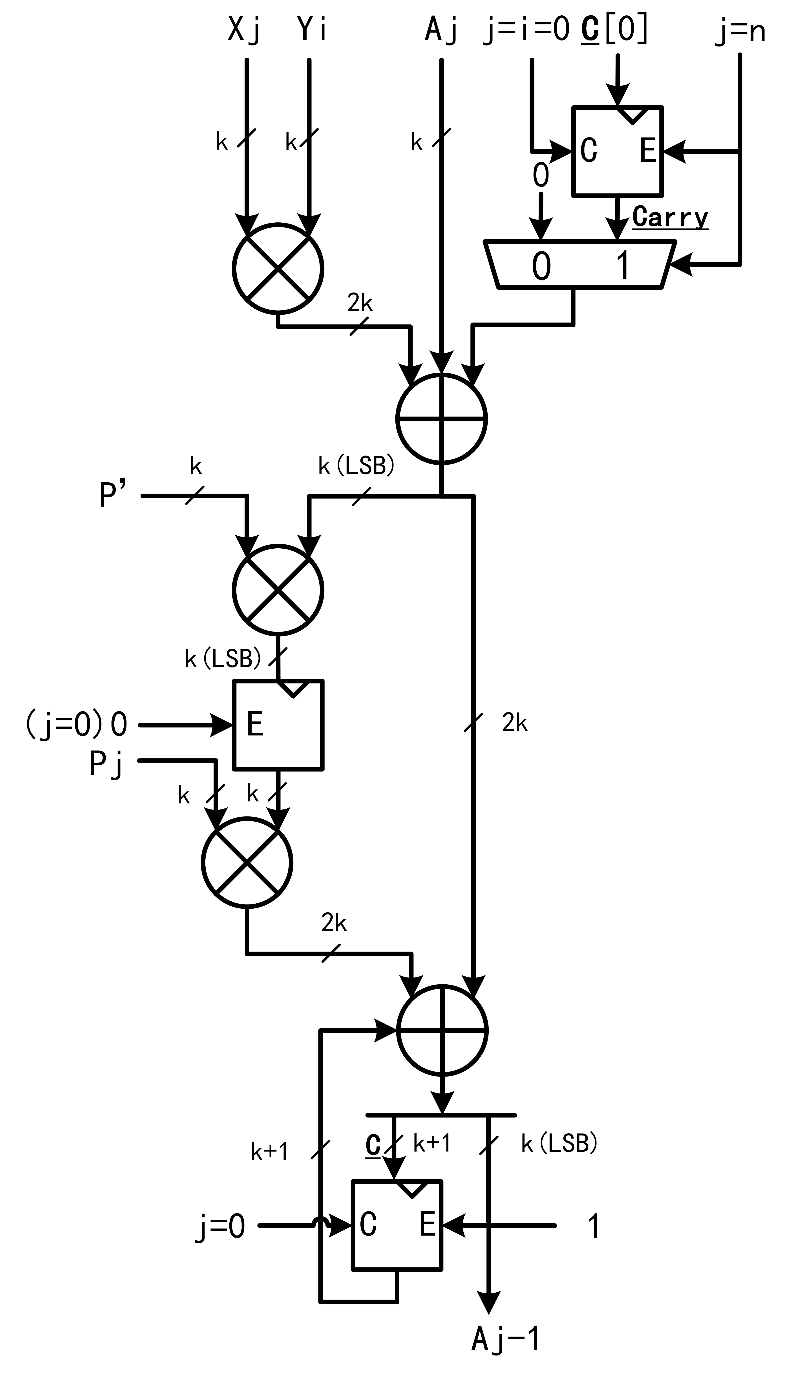
图4 本设计采用的蒙哥马利域模乘算法

### Process element原理

我们依照图4的IDDMM算法进行硬件设计。设计语言使用Verilog，最终在FPGA上实现相应功能。IDDMM算法的核心在于两层嵌套的FOR循环，硬件使用Process element模块进行实现。i从0到31遍历、j从0到32遍历。当j等于0时，需要根据S更新q，q再用于接下来的运算，这就需要再上下组合逻辑中插入一级寄存器，用来零时保存q值，这导致j等于0时，时序逻辑需要2个时钟计算出Uj。当j等于n时，S需要额外加上carry，这可以通过j等于n时选通多路选择器实现，carry可以在j等于n时保存。当j大于0时需要将运算结果保存到外部存储器A中。j从0到32遍历过程中，需要同步将结果暂存到寄存器carry中。在第16行，可以发现计算C需要用到上一次的结果，因此该加法器难以加入流水线，在本设计中使用DELAY2架构使得该加法器可以插入一级流水线。

遍历完i，j后，在RAM A。A的深度是33，宽度为128比特，最后一个地址存放的是carry。最后一步是读出A和模值p，对它们比大小。可以从高字开始比较，最长需要消耗32个时钟。比较完毕后，可能需要做减法，该减法可以通过取反加一后通过加法实现，128比特分组加法可以无时延的输出，但该加法需要从低字开始运算。最终结果会从低字开始输出，无论是否需要减法，输出会消耗32个时钟。这里为了时序收敛，减法器也使用了DELAY2架构，在需要做减法时，输出需要消耗64个时钟。

Process element模块硬件架构如图5所示。

图5 Process element (PE)

### Process element流水线实现与控制模块

由于PE中存在寄存器，流水线时序复杂，为了适配不同的流水线级数，外部流水控制器需要应对多种情况。根据图5可以发现，外部流水控制器的功能有以下3点：

1. 遍历i和j，从RAM取数据，输入到PE中的X、Y、P中。
2. 控制PE中3个寄存器的使能与清零。
3. 控制减法模块的运行。

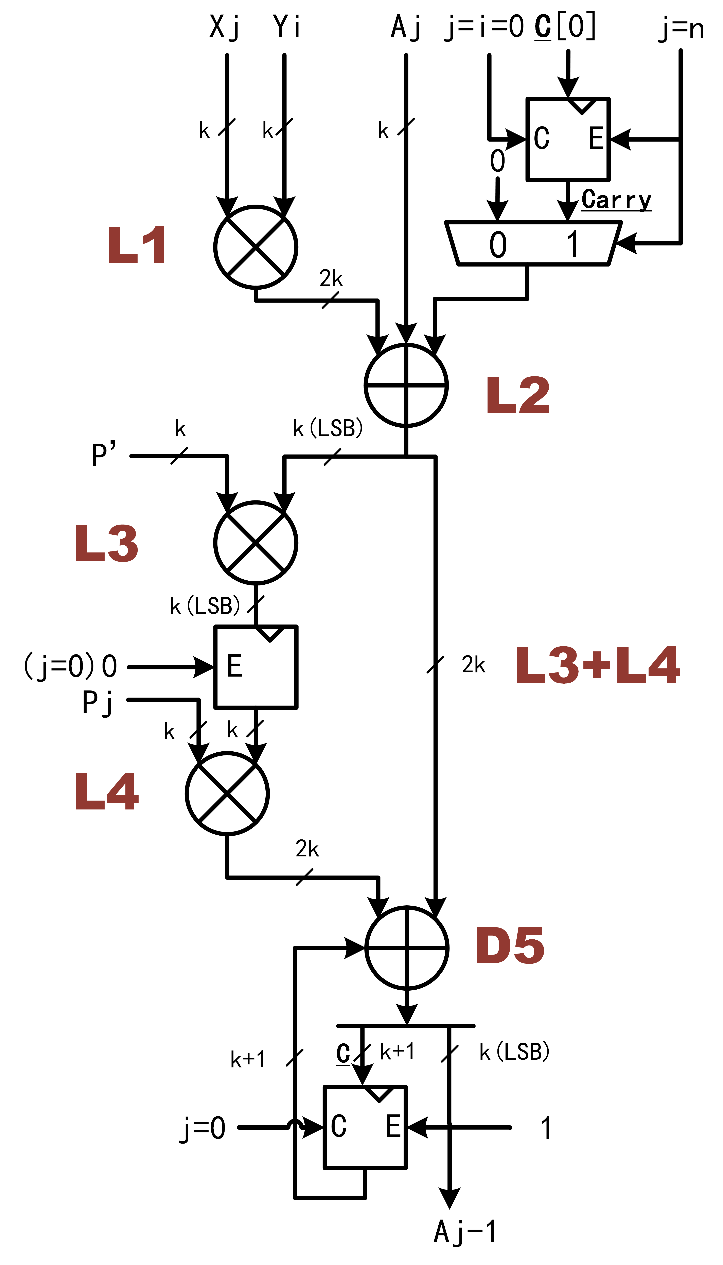
流水控制器需要知道每一个加法器或乘法器的流水线级数。用于产生i、j的时序。在图6中，流水线级数用Lx进行标记，在Verilog代码中部分parameter使用了与下图相同的名字。

图6 流水线级数Lx

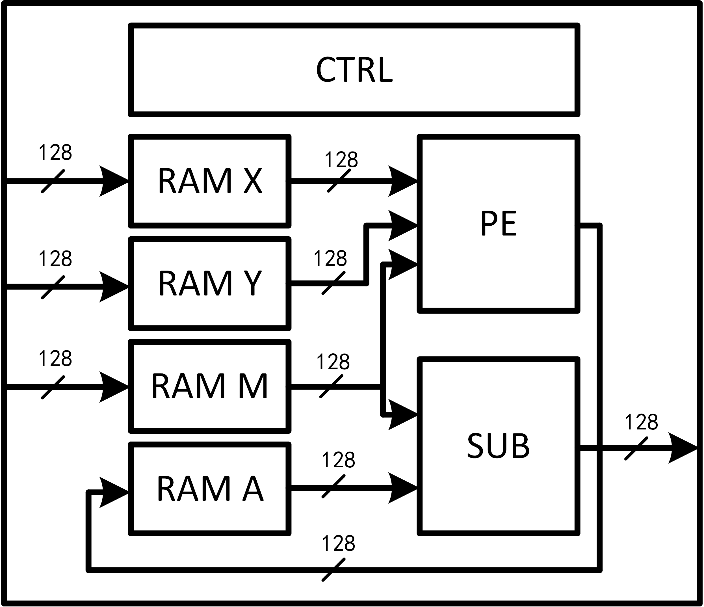
在图6中，当D5等于1时，就说明时序控制采用DELAY2架构。DELAY2架构的运算周期是原来的两倍，但时序更易收敛，因为D5所示的加法器可以插入一级流水线。在最终采用的设计中，每个乘法器流水线级数等于10，前加法器流水线级数等于2，后加法器流水线级数等于1。总共33级流水线。

图7 顶层图示

顶层视图如图7所示。其中RAM使用Verilog进行描述，4个RAM实际使用的深度不相同，其中X的深度是33，其余皆为32。为了读出第33个数据，在Y、M、A中使用了数据选择器减少了RAM的利用率。

### 仿真时序说明

顶层模块操作时序可通过查看mmp\_iddmm\_sp\_tb的仿真波形了解。通过定义宏\_VIEW\_WAVEFORM\_来查看单次波形。取消该定义，可以进行持续的随机测试。在顶层模块mmp\_iddmm\_sp.v中，用户接口如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位宽 | 方向 | 备注 |
| wr\_ena | 1 | input | 写使能 |
| wr\_addr | 5 | input | 写地址 |
| wr\_x | 128 | input | 写数据x |
| wr\_y | 128 | input | 写数据y |
| wr\_m | 128 | input | 写数据m |
| wr\_m1 | 128 | input | 写数据m’ |
| task\_req | 1 | input | 任务请求 |
| task\_end | 1 | output | 任务结束 |
| task\_grant | 1 | output | 输出有效 |
| task\_res | 128 | output | 输出数据 |

用户首先向RAM写入需要运算的数据，其中m1在写使能的第一个时钟被写入到模块，写使能期间需要保持m1。随后用户置高task\_req信号直到task\_end后再将它置低。task\_grant会在task\_req的后期置高，用户根据task\_grant信号将运算结果从task\_res读出。

本设计中，运算输出周期与运算方法设置有关。乘法器和加法器可以设置多种不同的实现方式，当后加法器设置为3-2\_DELAY2模式时，此时实现方法被叫做DELAY2架构。本设计在DELAY2架构的基础上，完全符合赛题三的指标。

|  |  |  |
| --- | --- | --- |
| 乘法器实现方法 | 前加法器实现方法 | 后加法器实现方法 |
| COMMON | COMMON | COMMON |
| TRADITION | 3-2\_PIPE1 | 3-2\_DELAY2 |
| VEDIC8 | 3-2\_PIPE2 | / |

### 指标

测试工具Vivado 2017.4 xc7k325t FPGA，在使用IDDMM 4096bit DELAY2架构，乘法器使用TRADITION方法，前加法器使用3-2\_PIPE2方法时，时序收敛时的时钟频率达到了304MHz，建立时间裕量0.008纳秒。最大消耗周期2293 cycles，最小消耗周期2261 cycles。RAM总共消耗16Kbit。LUT消耗5898个，Flip Flop消耗8486个，Xilinx DSP消耗164个。LUT等效ASIC GATE消耗147216门。

### 文件说明

├─doc 说明文档

├─sim iverilog或modelsim仿真

│ ├─mmp\_iddmm\_addend\_tb 后加法器仿真

│ ├─mmp\_iddmm\_addfirst\_tb 前加法器仿真

│ ├─mmp\_iddmm\_pe\_tb PE仿真

│ ├─mmp\_iddmm\_sp\_tb 顶层仿真

│ │ │ run\_iverilog.bat 运行iverilog仿真，用于随机验证或波形查看

│ │ │ run\_modelsim.bat 运行modelsim仿真

│ │ ├─rtl\_work modelsim文件

│ │ └─work modelsim文件

│ ├─mm\_iddmm\_pe\_tb 非流水线版本PE仿真，最终版本未用到

│ ├─mm\_iddmm\_sp\_tb非流水线版本顶层仿真

│ ├─mm\_iddmm\_sub\_tb减法器仿真，非流水线版本和流水线共用此文件

│ ├─mm\_r2mm\_2n\_tb 基2蒙哥马利算法仿真，用来与IDDMM对比。

│ ├─simple\_cclaa\_x4bit\_tb x4比特超前进位加法器仿真

│ ├─simple\_claa\_4bit\_tb 4比特超前进位加法器仿真

│ ├─simple\_mlclaa\_16bit\_tb 16比特超前进位多层级联加法器仿真

│ ├─simple\_vedic\_16bit\_tb 16比特VEDIC乘法器仿真

│ ├─simple\_vedic\_32bit\_tb 32比特VEDIC乘法器仿真

│ ├─simple\_vedic\_4bit\_tb 4比特VEDIC乘法器仿真

│ ├─simple\_vedic\_64bit\_tb 64比特VEDIC乘法器仿真

│ └─simple\_vedic\_8bit\_tb 8比特VEDIC乘法器仿真

├─src Verilog与Python模型代码

│ │ a0.mem 初始化RAM A内容

│ │ m.mem 初始化RAM M内容

│ │ mmp\_iddmm\_addend.v IDDMM前后法器模块

│ │ mmp\_iddmm\_addend\_tb.v IDDMM后加法器模块testbench

│ │ mmp\_iddmm\_addfirst.v IDDMM前加法器模块

│ │ mmp\_iddmm\_addfirst\_tb.v IDDMM前加法器模块testbench

│ │ mmp\_iddmm\_ctrl.v IDDMM控制器模块

│ │ mmp\_iddmm\_mul128.v IDDMM乘法器模块

│ │ mmp\_iddmm\_pe.v IDDMM PE

│ │ mmp\_iddmm\_pe\_tb.v IDDMM PE testbench

│ │ mmp\_iddmm\_shift.v IDDMM移位寄存器

│ │ mmp\_iddmm\_sp.v IDDMM顶层

│ │ mmp\_iddmm\_sp\_tb.v IDDMM顶层testbench

│ │ mmp\_iddmm\_top.v IDDMM FPGA顶层

│ │ mm\_iddmm\_pe.v 非流水线版本PE，未用到

│ │ mm\_iddmm\_pe\_tb.v 非流水线版本PE testbench，未用到

│ │ mm\_iddmm\_sp.v 非流水线版本顶层，未用到

│ │ mm\_iddmm\_sp\_tb.v非流水线版本顶层testbench，未用到

│ │ mm\_iddmm\_sub.v减法器模块，非流水线版本和流水线共用此文件

│ │ mm\_iddmm\_sub\_tb.v减法器模块testbench，未用到

│ │ mm\_iddmm\_top.v非流水线版本FPGA顶层，未用到

│ │ mm\_r2mm.v 基2蒙哥马利算法模块

│ │ mm\_r2mm\_2n.v基2蒙哥马利算法模块顶层

│ │ mm\_r2mm\_2n\_tb.v基2蒙哥马利算法模块顶层testbench

│ │ mm\_v\_sim\_model.v sv蒙哥马利大数计算模型，未用到

│ │ montgomery\_mul\_hd.py IDDMM算法Python模型

│ │ src\_impl 可综合源文件目录

│ │ src\_sim 可仿真源文件目录

│ │ x.mem 初始化RAM X内容

│ │ y.mem 初始化RAM Y内容

│ ├─bkup 备份，未用到

│ ├─common 通用件

│ │ └─mult32x32

│ │ │ mult.v IDDMM乘法器实现

│ │ ├─bk16x16-9备份，未用到

│ │ └─bk32x32-3备份，未用到

│ └─trash备份，未用到

├─vivado-mm非流水线版本vivado 2017.4工程，未用到

└─vivado-mmp IDDMM水线版本vivado 2017.4工程，最终成品