

Radix-8 Booth 译码 Montgomery 模乘的 RSA 算法的设计和硬件实现

张鑫, 王金城, 孙岩, 金西

(中国科技大学 物理系 微电子学教研室, 安徽 合肥 230026)

E-mail: jinxi@ustc.edu.cn

摘要: 提出一种使用 Radix-8 Booth 译码的 Montgomery 模乘算法, 进一步减少了模乘的中间乘积项个数, 提高了模乘的速度, 并给出基于该模乘算法的 1024 位 RSA 加密硬件的实现方案, 其加密速度可达到采用普通 Montgomery 模乘的 RSA 加密方案的 2 倍。在设计方法上使用基于系统级算法的快速设计流程, 在系统级设计阶段确定模乘和 RSA 整体算法的实现方案, 并对其评估及优化, 缩短了 RTL 阶段的设计时间, 加快了设计思想到硬件实现的转化。实现方案在自行设计的 FPGA 开发板上通过验证, 并进一步转换为 ASIC 设计综合。

关键词: Radix-8 Booth 译码; Montgomery 模乘; RSA 加密/解密

中图分类号: TP303

文献标识码: A

文章编号: 1000-1220(2008)05-0976-04

Design and Implementation of Radix-8 Booth-encoded Montgomery Modular Multiplication Algorithm for RSA Cryptosystem

ZHANG Xin, WANG Jin-cheng, SUN Yan, JIN Xi

(Microelectronics Laboratory Department of Physics, University of Science and Technology of China, Hefei 230026, China)

Abstract A Radix-8 Booth-encoded Montgomery modular multiplication algorithm is presented. Using this algorithm, iteration number is reduced to about 1/3 in each modular multiplication operation. Finally, we propose a 1024 bits RSA E/Decryption architecture based on this algorithm applying the system level modeling methodology, and verified on FPGA. The speed of the proposed algorithm is approximately 2 times of the most RSA VLSI designs based on original Montgomery modular multiplication algorithm.

Key words: radix-8 booth-encode technique; montgomery modular multiplication algorithm; RSA E/Decryption

1 引言

目前对 Montgomery 模乘算法的优化重点是减少部分积生成的个数和完成部分积相加的加法器造成的延时。使用 CSA (Carry-save Adder, 进位保留加法器) 可以将迭代中每个加法器延时减少为一级全加器延时。使用 Radix-4 Booth 译码 (基为 4 的 Booth 译码) 可以将迭代次数 n 缩短为原来的 1/2。文献 [1-4] 分别给出了在这些方面的改进。本文将在文献 [3] 和文献 [4] 的基础上, 给出一种新的使用 Radix-8 booth 译码和 CSA 的改进 Montgomery 模乘算法, 将迭代次数 n 缩短为原来的 1/3。

2 基于系统级算法的快速成型设计方法

为了便于快速地仿真验证 RSA 系统, 实现算法到 RTL 级设计的快速转化, 我们利用 SystemC 语言建立模乘和模幂的系统级模型, 并进行仿真、验证和优化。在 SystemC 设计阶段着眼点是 RSA 算法的实现, 对 RSA 算法的硬件实现的结构做

初步的划分, 为后续设计打好基础。在 RTL 级设计阶段, 借鉴系统级设计的结果, 采用更底层的优化方法和体系结构 (如流水线、CSA 等)。下文为了简化描述, 仍然使用伪 C 代码来描述各相关算法。

3 主要算法

3.1 Radix-8 booth 译码算法

对于一个 n 位二进制补码表示的数 A 和 B 的乘法 $A * B$, 数 B 可以表示成:

$$\begin{aligned} B &= -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \\ &= 2^{n-3}(-4b_{n-1} + 2b_{n-2} + b_{n-3} + b_{n-4}) \\ &+ 2^{n-6}(-4b_{n-4} + 2b_{n-5} + b_{n-6} + b_{n-7}) \\ &+ \dots \\ &+ 2^{3r-3}(-4b_{3r-1} + 2b_{3r-2} + b_{3r-3} + b_{3r-4}) \\ &+ \dots \\ &+ 2^0(-4b_2 + 2b_1 + b_0 + b_{-1}) \end{aligned} \quad (1)$$

收稿日期: 2007-01-15 收修改稿日期: 2007-09-13 作者简介: 张鑫, 男, 1983 年生, 硕士研究生, 研究方向为微电子学与固体电子学, SoC 芯片设计; 王金城, 1980 年生, 硕士, 研究方向为微电子学与固体电子学, VLSI 芯片设计; 孙岩, 男, 1982 年生, 硕士研究生, 研究方向为微电子学与固体电子学, VLSI 芯片设计; 金西, 男, 1970 年生, 副教授, 研究方向为微电子学与固体电子学, VLSI CAD、计算机应用

式(1)中 b_i 为B的第*i*位 对A * B 中的B 按照 $(-4b_{3i+1} + 2b_{3i+2} + b_{3i+3} + b_{3i+4})$ 的值重新编码, 部分积如表 1 所示, 参见文献[5]

表 1 Radix-8 Booth 译码对照表

Table 1 Radix-8 booth encoding truth table

$b_{3i-1}b_{3i-2}b_{3i-3}b_{3i-4}$	部分积 P_i	$b_{3i-1}b_{3i-2}b_{3i-3}b_{3i-4}$	部分积 P_i
0000	0A	1000	-4A
0001	+ 1A	1001	-3A
0010	+ 1A	1010	-3A
0011	+ 2A	1011	-2A
0100	+ 2A	1100	-2A
0101	+ 3A	1101	-1A
0110	+ 3A	1110	-1A
0111	+ 4A	1111	0A

则对于任何一个长度为 $n = 2^{2m}$ (m 为整数)位的B, 乘积A * B 都可表示为:

$$A * B = \sum_{i=0}^{n-1} A * b_i = \sum_{k=0}^{(n-1)/3} A * \text{Booth8-encode}(b_{3k+2}, b_{3k+1}, b_{3k}, b_{3k-1}) \quad (2)$$

其中 $b^{-1} = 0, b^{n+1} = b^n = 0$

这样就将相加的 n 个部分积减少为 $\lfloor n/3 \rfloor + 1$ 个. 而 Radix-4 booth 译码将 n 个部分积减少为 $n/2 + 1$ 个. 相比较而言 Radix-8 booth 译码与 Radix-4 booth 在设计上复杂程度相当, 而迭代次数大大减少. 特别值得关注的是: Radix-8 booth 译码 Montgomery 模乘与 Radix-4 booth 译码 Montgomery 模乘的关键路径的延时相差不大, 对于 RSA 这类涉及大数的计算将更利于芯片实现. 下面给出 Radix-8 booth 译码 Montgomery 模乘的具体实现

3.2 Radix-8 booth 译码 Montgomery 模乘的算法

Radix-8 Booth 译码的 Montgomery 模乘算法完成 $A * B * 2^m \text{ mod } N$ 的计算. 该算法的关键问题在于给出一个与模数 N 和 $A * B$ 有关的 s , 使得当前部分积 $(N * s + A * b_i)$ 的最低三位为零. 这样就可以通过对当前部分积右移三位实现其与下一个部分积的对齐操作. Booth 译码涉及到二进制补码表示的负数形式, 因此在部分积相加时应考虑符号扩展. 使用进位保留加法器完成部分积相加可以减少关键路径延时. 同时这也带来一个新的有关最低位进位的问题. 下面将给出 Montgomery 模乘算法的伪 C 代码和其中关键问题的处理方法

改进 Montgomery 模乘以及其中的 Booth 译码和 s 生成单元的伪 C 代码描述分别如下所示

```

Booth8MM (A, B, N)
{
  P[0]c = 0; P[0]s = 0;
  for(i = 0; i <= (n-1)/3; i++)
  {
    Q[i] = P[i]/8 + Booth(b3i+2, b3i+1, b3i, b3i-1) * A; ... (1)
    qi = (Q[i][2:0] + Ci) mod 8; ... (2)
    P[i+1] = Q[i] + Ci + S-Gen(qi, n2, n1) * N; ... (3)
    Ci = (P[i+1]sum[2] or P[i+1]sum[1]); ... (4)
  }
}

```

```

Result = P[(n-1)/3 + 1]carry << 1 + P[(n-1)/3 + 1]ssum
return Result;
}

```

```

Booth(b2i+2, b2i+1, b2i, b2i-1)
{
  switch({b2i+2, b2i+1, b2i, b2i-1})
  {
    case 0: c = 0; break;
    case 1: c = 1; break;
    case 2: c = 1; break;
    case 3: c = 2; break;
    case 4: c = 2; break;
    case 5: c = 3; break;
    case 6: c = 3; break;
    case 7: c = 4; break;
    case 8: c = -4; break;
    case 9: c = -3; break;
    case 10: c = -3; break;
    case 11: c = -2; break;
    case 12: c = -2; break;
    case 13: c = -1; break;
    case 14: c = -1; break;
    case 15: c = 0; break;
    case 16: c = 1; break;
  }
}
return c;
}

```

```

S-Gen(qi, n2, n1)
{
  s0 = qi[0];
  switch({n2, n1})
  {
    case 0:
      {
        s1 = qi[1] ⊕ qi[0];
        s2 = qi[2] ⊕ qi[1] ⊕ qi[0] ⊕ (qi[1] & qi[0]);
      }
    case 1:
      {
        s1 = qi[1];
        s2 = qi[2] ⊕ qi[0];
      }
    case 2:
      {
        s1 = qi[1] ⊕ qi[0];
        s2 = qi[2] ⊕ qi[1] ⊕ (qi[1] & qi[0]);
      }
    case 3:
      {
        s1 = qi[1];
        s2 = qi[2];
      }
  }
}
return {s2, s1, s0};
}

```

4 RSA 模幂的算法

设输入数据为 $X = \sum_{i=0}^{n-1} x_i 2^i$, 加密密钥为 $E = \sum_{i=0}^{n-1} e_i 2^i$, 模为 N

$$= \sum_{i=0}^{n-1} n_i 2^i$$

RSA 模幂公式可展开为 n 次模乘的形式:

$$X^E = X^{e_0} X^{2e_1} \dots X^{2^{n-1} e_{n-1}} \quad (3)$$



RSA 模幂模块由Booth8MM 模乘模块和控制逻辑等构成 首先进行Booth8MM (X, 1, N)将输入数据X 映射成 $X' = X * 2^m \pmod N$, 然后采用 X' 进行模幂运算得到 $X^E * 2^m \pmod N$. 然后再次调用模乘模块完成Booth8MM ($X^E * 2^m \pmod N, 1, N$), 去掉模幂运算中引入的因子 2^m , 最后得到结果 $X^E \pmod N$. 下面给出了模幂的伪C 代码描述

```

Booth8ME (X, E, N)
{
  Nr = Booth8MM (2n, 2n, N) = 2n mod N (1)
  Z0 = Booth8MM (Nr, X, N) (2)
  for (i = 0; i < n-1; i++)
  {
    {Zi+1 = Booth8MM (Zi, Zi, N) (3)
    if (ei = 1)
      Zi+1 = Booth8MM (Zi+1, Z0, N) (4)
    }
    Zn-1 = Booth8MM (Zn-1, 1, N) (5)
  }
  return Zn-1
}

```

其中Nr 和 Z0 的计算在硬件实现时, 并不是植入新的Booth8MM 单元, 而是嵌入到循环中的Booth8MM 单元中计算的, 即整个模幂运算只有一个模乘单元

5 Radix-8 booth 译码 Montgomery 模乘的实现

5.1 部分积的生成

表1中2A、4A 通过对A 移位得到, 二进制补码形式的-A、-2A、-4A 的生成方式是先对A、2A、4A 取反, 再以进位的形式对其加1. 3A 是通过得到的, 而-3A 是通过得到的. 具体实现是通过在图1中第一个(4, 2)压缩器构成的CSA 完成. 完成二进制补码转换所加的1 作为进位输入到CSA 中.

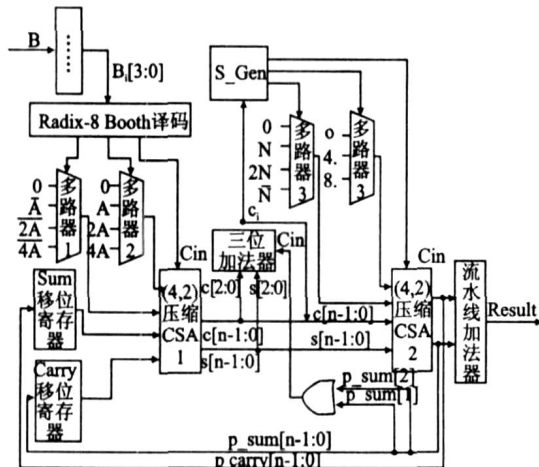


图1 Montgomery 模乘的模块内部结构

Fig 1 Montgomery modular multiplication module schematic

5.2 S.Gen 的实现

Booth8MM 算法(3)中S-Gen (q_i, n_2, n_1)的作用是产生上文所述的s. 为了使 $Q_{i+1} = S-Gen(q_i, n_2, n_1) * N$ 的最低三位为零, 需要选择合适的N 的倍数使二进制数 $Q_{i+1} = S-Gen(q_i, n_2, n_1) * N$ 为零(由于N 是奇数, n_0 为1). 令 $q_2q_1q_0 + s * (n_2n_1)$ 结果为0, 可以求解出s 的逻辑表达式. 该倍数的取值范围为0-7.

因此需要求出0, 1N, 2N, 3N, 4N, 5N, 6N, 7N. 其中2N、4N、8N 通过对N 移位得到. 3N、5N、6N 分别通过在第二个CSA 中完成 $2N + N, 4N + N, 4N + 2N$ 得到. 7N 通过得到, 所加的1 作为进位输入到图1中(4, 2)压缩器CSA 2 中.

5.3 符号扩展

由于Booth 译码涉及到二进制补码计算, 因此在迭代中每次进行部分积相加时都要进行符号扩展. 即在Booth8MM 算法的第(3)步移位必须是算术右移, 左侧移入符号位. 这样就通过简单的逻辑实现了符号扩展.

5.4 进位信息的保留

注意到Booth8MM 算法的第(3)步使用CSA 得到的进位保留形式的中间结果和直接使用CPA (carry propagate adder 进位传递加法器)的中间结果移位情况不同. 使用CSA 加法器在右移三位的操作中, 最低三位的进位情况将会丢失. 例如最低三位为的情况, 在进位保留的形式下, 产生最高位进位之前已经对Carry 和Sum 进行了移位, 其最低三位的进位信息丢失了. 为完成正确的计算, 该进位的信息需要保留下来. 由于carry[0]总是零, 因此当sum [2]和sum [1]之一为1时carry [2: 0] + sum [2: 0]就会产生进位. 具体实现中只需使用一个输入为sum [2]和sum [1]的或门就可产生此进位.

5.5 部分积相加的实现

在Booth8MM 模乘算法中, 使用CSA 完成算法中(1)、(3)步的加法操作, 提高迭代速度. 这两个CSA 加法器使用4-2压缩器构成, 即图1中(4, 2)压缩器CSA 1和(4, 2)压缩器CSA 2. 它将比使用两级全加器直接构成的CSA 减少两个门延时, 参见文献[6].

5.6 最终结果的生成

最终结果加法器使用带流水线结构的进位传递加法器, 对于n 较大的模乘(如 $n \geq 1024$), 可以减少该加法操作占用的资源. 每次由一个32位CLA (Carry-look-ahead adder)

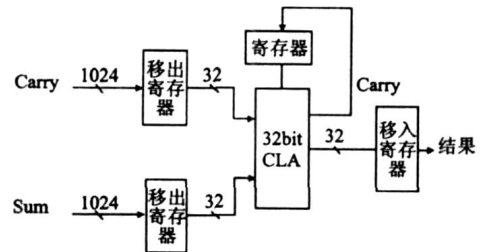


图2 带流水线结构的1024bits 进位传递加法器
Fig 2 The architecture of 1024 bits pipelined CPA

完成32 位加数与被加数的求和, 生成部分和该部分和的进位信息. 移出寄存器由多路器和寄存器构成, 每时钟周期移出32 位数据. 移入寄存器由移位器和寄存器构成, 每时钟周期移入32 位数据. 流水线加法器的结构如图2 所示.

6 RSA 模幂的硬件实现

6.1 硬件结构

图3 给出了1024 位的RSA 处理器结构框图. 整个系统由

改进 Montgomery 模乘模块和控制部分组成, 控制部分主要完成控制模乘模块的输入和输出功能 运算位宽可调整, 最大位宽是 1024 位, 但支持满足 RSA 密钥对条件的低于 1024

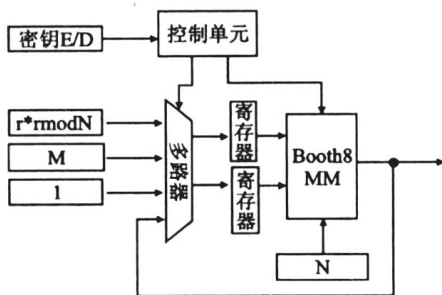


图3 RSA 加密/解密处理器结构框图

Fig 3 The architecture of proposed RSA processor

位的所有位宽的 RSA 运算, 如常见的 768 位、512 位、256 位等等 可以在加密过程中随时更改密钥对, 因为设计输入是 M、E、N 和 N_r , 即明文/密文、密钥、模 加密过程中改变 E、N 可以增加信息的保密程度, 解密时根据数据达到时间使用相应的解密密钥对, 即可解密

7 结果分析

加密密钥 E 为 n 位的 RSA 操作所需模乘次数可以按照如下公式计算:

$$\log_2 E + v(E) + 2 \tag{4}$$

其中 $v(E)$ 为 E 中 1 的个数 在 E 中 1 和 0 的个数相等的平均的情况, 模幂算法需要 $3n/2 + 2$ 次模乘运算 在 E 各位全为 1 的最坏情况下, 需要 $2n + 2$ 次模乘运算 一个完整的 Radix-8 Booth 译码的 Montgomery 模乘运算需要 $((n + 2)/3 + 32)$ 个时钟周期, 其中引入的 32 是由于采用了流水线加法器完成模乘最后结果转换 表 2 给出了 n 位 RSA 加/解密操作所需时钟周期数

表 2 n 位 RSA 加/解密操作所需时钟周期

Table 2 Time cost in n bits RSA processes

状况	所需时钟周期(cycle)
平均状况	$(3n/2 + 2) * ((n + 2)/3 + 32)$
最差状况	$(2n + 2) * ((n + 2)/3 + 32)$

我们在具体实现中采用 1024 位加/解密 平均况下需要 575k 个时钟周期完成一次 RSA 操作, 最坏情况下需要 766k 个时钟周期完成一次 RSA 操作 验证系统为 BGA 封装具有 80 万等效门的 Xilinx XCV 800-4BG560 FPGA, 系统时钟由开发板上晶振提供 在 20Mhz 时钟频率下, 每秒钟可进行 24 次 1024 位加密运算 实际消耗 23.03% 的时序逻辑资源, 消耗 59.68% 的组合逻辑资源

使用 Synopsys 公司的 DC 和 TSMC 13 工艺库综合后,

在宽松的面积约束下保守的系统时钟频率可以达到 270MHz 在 270MHz 工作频率下, 对 1024 位的 RSA 操作, 其加密速度平均为 481 kbit/s, 在最坏情况下为 361 kbit/s

在 200MHz 工作频率的同等情况下, 加密速度平均为 356 kbit/s, 在最坏情况下为 267 kbit/s, 而文献 [4] 加密速度平均为 238 kbit/s, 在最坏情况下为 178 kbit/s, 可以看出 Radix-8 Booth 译码具有较大的优势

8 结束语

从减少模乘的中间乘积项入手提出了一种新的使用 Radix-8 Booth 译码的 Montgomery 模乘算法, 进一步提高了 Montgomery 模乘的速度; 并使用文献 [7] 给出的基于系统级算法的快速设计流程, 使用 SystemC 语言在系统级设计阶段将模乘和 RSA 的算法快速转化为实现方案, 并直接进行初步评估及优化, 加快了设计思想到硬件实现的转化; 在 RTL 设计阶段对其硬件设计进行了低层次的优化, 进一步提高了解密速度; 给出了保守时钟频率 270MHz, 解密速度为 361 kbit/s 的 1024 位 RSA 加/解密的硬件实现

References

- [1] Yang C C, Chang T S, Jen C W. A new RSA cryptosystem hardware design based on montgomery's algorithm [J]. IEEE Trans on Circuits and System-II, 1998, 45(7): 908-913
- [2] Kim Y S, Kang W S, Choi J R. A synchronous implementation of 1024bit modular processor for RSA cryptosystem [C]. AP-ASIC2000 Proceedings of the Second IEEE Asia Pacific Conference New York: IEEE, 2000, 187-190
- [3] Leu J J. Design methodology for booth-encoded montgomery module design for RSA cryptosystem [C]. ISCAS 2000 IEEE Int Sym on Circuits and Systems New York: IEEE, 2000, 57-360
- [4] Shu Yan, Lu Junming VLSI implementation of RSA cryptosystem based on the booth-encoded montgomery module [J]. Journal Of Xidian University, 2002, 29(3): 363-367
- [5] Hidalgo J A. A radix-8 multiplier unit design for specific purpose [C]. XIII Conference of Design of Circuits and Integrated Systems (DCIS '98). Madrid 1998
- [6] Reto Zimmemann Binary adder architectures for cell-based VLSI and their synthesis [D]. Swiss Federal Institute Of Technology. Diss. ETH No. 12480
- [7] Wang Jin-cheng, Deng Bo-ren, Jin Xi A fast flow of chip-design Based on system level modeling methodology [J]. Micro-computer & Its Applications, 2005, 24(3): 24-26

附中文参考文献:

- [4] 舒 妍, 卢君明 基于 Booth 编码模乘模块的 VLSI 设计 [J]. 西安电子科技大学学报, 2002, 29(3): 363-367
- [7] 王金城, 邓博仁, 金 西 一种基于系统级算法的芯片快速成型设计流程 [J]. 微型机与应用, 2005, 24(3): 24-26