REDC3329 设计文档

李远政 U202211014

2025 年 9 月

目录

1 模块要求

根据任务要求实现一个 12-bit 位宽的流水线快速模乘器模块。模乘器输入 a、b 两个 12-bit 位宽的乘数，给定模数 q=3329，在固定周期内输出 a*b mod q。模块 IO 接口如表 1 所示，要求当模乘器输入使能有效时，a、b 两个乘数输入，随后模乘器进行计算，在模乘器计算过程中，busy 信号指示高电平为忙状态，否则为低电平指示空闲状态。在经过固定周期后，done 信号由低变高，指示计算完成，同时输出结果。

2 模块端口说明

表 1: 实验所需的元器件

端口	方向	位宽	描述
`clk`	Input	1	系统时钟信号。
`rst_n`	Input	1	低电平有效的异步复位信号。
`en`	Input	1	使能信号。当 `en` 为高时，模块在下一个时钟周期锁存输入 `a` 和 `b`。
`a`	Input	12	输入操作数 A。
`b`	Input	12	输入操作数 B。
`busy`	Output	1	忙信号。当模块正在处理数据时（从 `en` 拉高到 `done` 拉高之间），该信号为高。
`done`	Output	1	完成信号。当一次模乘计算完成，且输出 `r` 有效时，该信号拉高一个时钟周期。
`r`	Output	12	计算结果，即 `(a * b) mod 3329`。

3 算法理论与算法验证

3.1 Montgomery 模乘算法

可以观察到题目中所给出的 3329 是一个质数，所以可以使用蒙哥马利算法简化取模的计算过程。算法的基本原理是通过一系列的转换将本来对一个普通数 N 的取模转换为对一个 2 的幂次 R 的取模过程，从而可以通过移位算法节省大量计算资源。为了达到对一个数 T 转换模数的目的，先构造 REDC(a,b,R) 函数解决 $(ab) \cdot R^{-1}(modN)$ 的问题, 这个函数是存在简便算法的。然后反复调用这个函数来实现模乘的过程。具体步骤如下:

3.1.1 REDC 函数的设计

构造函数的基本思路是将大数 T 加上一个适当的数值 m，使其能够被 R 整除，并且结果能够落在 0 到 2N 的范围内，并且仍然保证对 N 同余的特性。这样就可以直接使用简单的数据选择器实现对于 N 的取模。具体步骤如下：首先构造加数:

$$m = N[T(modR)N'](modR)$$

其中:

$$N \cdot N' = -1(modR)$$

相加之后得到 t：

$$t = T+m = T+N[T(modR)N'](modR) = T+[(NN')(modR)][(T(modR))](modR) = T-T(modR)$$

将 t 再 modR 的情况下进行化简：

$$t(modR) = T + [(NN')(modR)][(T(modR))](modR) = T - T(modR)$$

从上面的变换中，不难看出这个 t 是可以被 R 整除的。接下来将 t 除以 R 得到：

$$u = \frac{T+m}{R}$$

现在来估算 u 的范围：

$$T < N \cdot R$$

$$N[T(modR)N'](modR) < N \cdot R$$

$$0 \leq t < 2N$$

因此可以直接通过比较 t 和 N 的大小来实现对 N 的取模：

3.1.2 REDC 函数的乘法性质

因为直接调用 REDC 函数会带有 R 的逆元，无法直接得到结果，所以通过多次调用转换来实现模乘的过程。具体步骤如下：

1. 转换输入 a,b 为蒙哥马利域下的数：

$$a_{mont} = REDC(a, R^2)$$

$$b_{mont} = REDC(b, R^2)$$

2. 计算 a,b 的乘积：

$$t = REDC(a_{mont}, b_{mont})$$

3. 将结果转换回普通域下的数：

$$r = REDC(t, 1)$$

这样就刚好通过两部约去了最开始引入的 R 的次方影响，得到了正确的结果。

4 电路设计

4.1 流水线结构设计

完成一次 REDC 操作一共需要调用三次乘法单元，分别是两输入数据相乘产生 T，T 与 N' 相乘与 R 取模产生 m，m 与 3329 相乘产生最终的加数。而完成一次模乘操作需要调用四次 REDC 单元。所以如果使用组合逻辑完成 REDC 操作，则在完成一次乘法操作的过程中会导致另外两组乘法电路的空闲，降低了电路的利用率，因此可以通过设计合理的流水时序，将一个 REDc 模块拆解成三个乘法部分进行流水化改造，从而提高电路的利用率。

表 2: REDC 模块调用输入参数

调用	data1	data2
$REDC(a, R^2)$	a	R^2
$REDC(b, R^2)$	b	R^2
$REDC(a_{mont}, b_{mont})$	a_{mont}	b_{mont}
$REDC(t, 1)$	t	1

4.2 REDC 模块的拆分

4.3 流水时序设计

下面通过表格列出四次调用 REDC 模块时，输入参数的数据来源：观察表格，可以发现两个优化的方向。

1. 在保留一条流水线的情况下，四次调用之间存在很强的前后依赖关系，a_{mont} 和 b_{mont} 的计算需要先后完成才能够进行第三次的调用，这会导致流水线存在单元空闲等待的情况发生，因此在设计时序的时候可以在 a、b 计算的时候插入其他数据组的计算。

2. 在最后一次调用中，data2 恒为 1，因此可以直接跳过第一步的乘法单元，直接接入 REDC 的第二部分进行计算，从而节省出来了一个时钟周期的乘法单元的调用。

用 A、B、C、D 分别表示对于一组 a、b 输入数据的四次调用，用下标区分不同组的输入数据。其中 C 对于 A、B 步骤存在先后依赖关系，D 对于 C 存在依赖关系并且 D 可以直接跳过 part1 开始计算，设计得到的流水线时序如下，下划线部分标识了一组数据的所有占用。:

表 3: 流水线时序设计

clkcnt	part1	part2	part3
1	$\underline{\boldsymbol{A_1}}$	D_{-1}	B_0
2	C_0	$\underline{\boldsymbol{A_1}}$	D_{-1}
3	$\underline{\boldsymbol{B_1}}$	C_0	$\underline{\boldsymbol{A_1}}$
4	$\times$	$\underline{\boldsymbol{B_1}}$	C_0
5	A_2	D_0	$\underline{\boldsymbol{B_1}}$
6	$\underline{\boldsymbol{C_1}}$	A_2	D_0
7	B_2	$\underline{\boldsymbol{C_1}}$	A_2
8	$\times$	B_2	$\underline{\boldsymbol{C_1}}$
9	A_3	$\underline{\boldsymbol{D_1}}$	B_2
10	C_2	A_3	$\underline{\boldsymbol{D_1}}$

观察流水线时序可以发现，通过存在等待的周期中插入其他数据组的计算，可以大幅度提升流水线的使用效率，从而使得整个流水线在大多数时候几乎都处于完全占用的状态。从时序表也可看出，对于第一个上升沿输入的数据，需要延迟到第 10 个时钟周期才能够输出结果。

5 模块验证

5.1 测试平台设计

使用 verilator 作为仿真工具，编写测试平台对设计的流水线模乘器进行功能验证。由于设计的模乘器位宽不高，因此可以直接使用穷举的方式完成验证。验证平台的设计简图如下：

6 FPGA 综合实现报告