# Fast Modular Multiplication

Yuval Domb
yuval@ingonyama.com

July 25, 2022 (version 1.0)

**Abstract**

Modular multiplication is arguably the most computationally-dominant arithmetic primitive in any cryptographic system. This note presents an efficient, hardware-friendly algorithm that, to the best of the author's knowledge, outperforms existing algorithms to date.

## 1 Introduction

The standard modulo-prime multiplication problem in $\mathbb{F}_s$ can be cast as

$$r = a \cdot b \mod s \tag{1}$$

where $a, b, r \in \mathbb{F}_s$, $s$ is prime, and the standard $\mathbb{Z}$-algebra is utilized. Equivalently this can be written as

$$a \cdot b = l \cdot s + r \tag{2}$$

with $l \in \mathbb{Z}$ such that $0 \leq r < s$.

The purpose of this note is to provide an efficient, hardware-friendly method for fast computation of (1).

Assume that all variables are represented by $d$-radix digits, and denote by $n$ the required number of digits used to represent any element in $\mathbb{F}_s$, thus

$$n = \lceil \log_d s \rceil \tag{3}$$

For simplicity, let us set $d = 2$ and use bits in place of digits, for the remainder of the note.

Finally, although this note is primarily concerned with modulo-prime multiplication, its results can be generalized to any case of $a \mod s$ where $a < s^2$ for any $s$, prime or not.

## 2 Contribution

The main contribution of this note is to show how Barrett's Reduction [1], together with good parameter selection and a simple bounding technique, can be used to approximate the quotient $l$ up to a small constant error independent of $n$, for any $n$. Surprisingly, the resulting reduction algorithm closely resembles Montgomery's Modular-Multiplication algorithm [2], without the coordinate translation requirement. This bounding technique can be used to further lower the calculation complexity of special cases of interest, not presented here, at the price of increased constant error.

# 3 Reduction Scheme

## 3.1 Assume $l$ is approximately known

Assume that $l$ is approximately known and denote by $\hat{l}$ its approximation, such that

$$l - \lambda \leq \hat{l} \leq l \tag{4}$$

where $\lambda = O(1)$ is a known constant.

Starting initially with $\lambda = 0$, it is clear that

$$ab[2n-1:0] - \hat{l}s[2n-1:0] = r[n-1:0] \tag{5}$$

where the brackets denote bit locations and sizes. Note that $\lambda = 0$ means that we know apriori that the remainder is at most $n$ bits in length, so that all remaining most-significant (ms) bits in the rhs of (5) must be zero.

Utilizing simple bit manipulation, this can be cast as the following long addition

$$\begin{array}{cccccccc} & & & & & & & 1 \\ & \overline{\hat{l}s[2n-1]} & \dots & \overline{\hat{l}s[n]} & \overline{\hat{l}s[n-1]} & \dots & \overline{\hat{l}s[0]} \\ + & ab[2n-1] & \dots & ab[n] & ab[n-1] & \dots & ab[0] \\ \hline & 0 & \dots & 0 & r[n-1] & \dots & r[0] \end{array}$$

where the over-line denotes the bit-inversion operator and the 1-bit at the top-right is treated as an initial carry bit. The important insight is that only $ab[n-1:0]$ and $\hat{l}s[n-1:0]$ are necessary in order to complete this calculation, which immediately saves approximately half of the calculations. Note that the resulting adder is a *fixed width adder*, (i.e. $n+n \to n$). This means that any overflow ms bits must be ignored. An equivalent alternative to the above is a fixed-width subtractor ($n-n \to n$), where the result is treated as an unsigned integer.

Let us denote the type of multiplier required to generate the above products as an $n \times n \to n_{\mathbf{lsb}}$ multiplier, where $n_{\mathbf{lsb}}$ refers to the $n$ least-significant bits of the full product. This means that the products $a \cdot b$ and $\hat{l} \cdot s$ can be generated using $n \times n \to n_{\mathbf{lsb}}$ multipliers. In addition, if $s$ is constant, $\hat{l} \cdot s$ can be generated using a constant $n \times n \to n_{\mathbf{lsb}}$ multiplier.

Finally when $\lambda \neq 0$

$$ab - \hat{l}s = r + \lambda s \tag{6}$$

and the number of bits required to represent the rhs of (5) is

$$\lceil \log_2(r + \lambda s) \rceil \leq n + \left\lceil \log_2 \frac{r + \lambda s}{s} \right\rceil \leq n + \lceil \log_2(1 + \lambda) \rceil \tag{7}$$

so if $\lambda = 1$ the total number of additional bits required would be 1.

## 3.2 Use Barrett's Reduction to approximate $l$

Barrett's modular reduction approximates $l$ as follows

$$l = \left\lfloor \frac{ab}{s} \right\rfloor = \lim_{k \to \infty} \frac{ab \cdot m(k)}{2^{k+n}} \tag{8}$$

where

$$m(k) = \left\lfloor \frac{2^{k+n}}{s} \right\rfloor < 2^{k+1} \tag{9}$$

is a function of the $k$, representing the maximal, $k+1$ bits, lower-bound approximator for (8). For finite $k$, the approximation error is

$$e(k) \equiv \frac{1}{s} - \frac{m(k)}{2^{k+n}} < 2^{-(k+n)} \tag{10}$$

where the upper-bound can be derived by examining the maximal difference between the binary representation of the left and right terms. This immediately leads to the approximation error on $l(k)$

$$e(l,k) \equiv \frac{ab}{s} - \frac{ab \cdot m(k)}{2^{k+n}} < 2^{2n} \cdot 2^{-(k+n)} = 2^{n-k} \tag{11}$$

Thus if $k \geq n$ the approximation error is at most 1.

### 3.3 Choose the parameters and bound the error

Choosing $k = n$ (i.e. $m(n) < 2^{n+1}$) leads to the following approximation on $l$

$$\hat{l}_0 = \left\lfloor \frac{abm}{2^{2n}} \right\rfloor \tag{12}$$

$$e(\hat{l}_0) < 1 \tag{13}$$

where the multiplication is $n \times n \times (n+1) \to (n+1)_{\mathbf{msb}}$, and the approximation error obeys (11).

Let us instead perform the above multiplication in two stages. Initially, assume that $ab[2n-1:0]$ is available and perform the multiplication as follows

$$\frac{abm}{2^{2n}} = \frac{ab[2n-1:n] \cdot m}{2^n} + \frac{ab[n-1:0] \cdot m}{2^{2n}} \tag{14}$$

$$< \frac{ab[2n-1:n] \cdot m}{2^n} + 2 \tag{15}$$

where the right-most term is trivially upper-bounded by 2. This immediately leads to the following approximation on $l$

$$\hat{l}_1 = \left\lfloor \left\lfloor \frac{ab}{2^n} \right\rfloor \cdot \frac{m}{2^n} \right\rfloor \tag{16}$$

$$e(\hat{l}_1) < 3 \tag{17}$$

where the inner multiplication is $n \times n \to n_{\mathbf{msb}}$, the outer (constant) multiplication is $n \times (n+1) \to (n+1)_{\mathbf{msb}}$, and the approximation error is upper-bounded by the sum of (13) and the right-most term of (15). Note that since $m(n)$ is typically very close to $2^n$ and $n$ is typically large, no additional bits need to be added (i.e. $n+1$ bits suffice) to represent the constant error (17). Nonetheless, this overflow corner-case must be examined and ruled out per a given setup.

### 3.4 Putting it all together

Below is a block diagram of the hardware-optimized modular multiplier. The diagram assumes that $s$ and $m$ are known constants, and uses the $\hat{l}_1$ approximation for $l$.

Note that the left-most multiplication module is independent of the reduction logic, allowing the remainder of the circuit to be generalized beyond multiplication reductions.

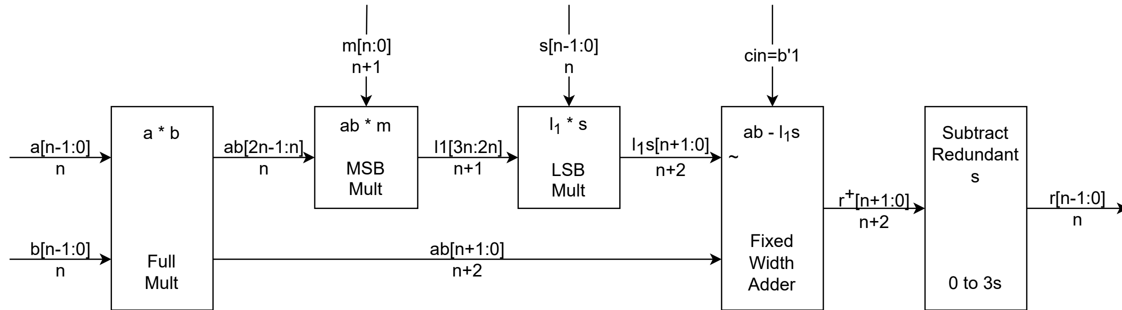


Figure 1: Hardware-optimized modular multiplier

### 3.5 Examples

#### 3.5.1 Example for n=16

```
— params —
n = 16
s[n−1:0] = 65521
m[n:0] = 65551
a[n−1:0] = 64111
b[n−1:0] = 11195

— direct calc —
l[n:0] = 10954
r[n−1:0] = 5611
5611 = 64111*11195 − 10954*65521

— a*b full mult —
ab[2n−1:0] = 717722645
ab[2n−1:n] = 10951
ab[n+1:0] = 234517

— ab*m msb mult —
l1[3n:2n] = 10953
e(l1) = 1

— l1*s lsb mult —
l1s[n+1:0] = 163385

— ab−l1s fixed width adder —
r+[n+1:0] = 71132
```

```
— subtract redundant s —
r_hat = 5611
```

### 3.5.2 Example for n=32

```
— params —
n = 32
s[n-1:0] = 4294967291
m[n:0] = 4294967301
a[n-1:0] = 1152833672
b[n-1:0] = 2546222476

— direct calc —
l[n:0] = 683444321
r[n-1:0] = 2821307461
2821307461 = 1152833672*2546222476 - 683444321*4294967291

— a*b full mult —
ab[2n-1:0] = 2935371006736011872
ab[2n-1:n] = 683444320
ab[n+1:0] = 3699053152

— ab*m msb mult —
l1[3n:2n] = 683444320
e(l1) = 1

— l1*s lsb mult —
l1s[n+1:0] = 13762647584

— ab-l1s fixed width adder —
r+[n+1:0] = 7116274752

— subtract redundant s —
r_hat = 2821307461
```

### 3.5.3 An exotic example

If $s = 65717$ and $a = 65535, b = 65631$, we get that the real value of $l$ is $\left\lfloor \frac{ab}{s} \right\rfloor = 65449$. On the other hand, our approximation gives $\hat{l}_1 = 65446$, and so the error $e(\hat{l}_1)$ is 3 in this case. However, primes $s$ for which such examples exist are sparse and for most primes, the largest possible error will not exceed 2.

## References


[1] Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology — CRYPTO' 86*, pages 311–323, 1987.

[2] Peter L. Montgomery. Modular multiplication without trial division. In *Mathematics of Computation*, volume 44, pages 519–521, 1985.