Comparison of Montgomery and Barrett modular multipliers on FPGAs

Yinan Kong and Braden Phillips
Centre for High Performance Integrated Technologies & Systems (CHiPTec)
School of Electrical & Electronic Engineering
The University of Adelaide, Adelaide
SA 5005, Australia

Abstract - **A diverse variety of algorithms and architectures for modular multiplication have been published. This paper concentrates on 2 algorithms, Montgomery and Barrett, and provides area and timing results for FPGA implementations of different architectures and wordlengths. The results show that techniques such as quotient pipelining and trivial quotient digit selection are not well suited to FPGA implementations, but that high-radix, separated modular multipliers perform well on this platform.**

Keywords - **computer arithmetic, modular multiplication, FPGA**

I. INTRODUCTION

Modular multiplication is important for many applications, including cryptography and image processing. In this paper we consider the practical performance of algorithms to compute the modular multiplication $C = A \times B \bmod N$ where A, B, C and N are all n-bit integers. We consider values of n up to 32-bits. Two types of modular multiplier are examined: those that follow Montgomery's algorithm [1]; and those based on Barrett's algorithm [2]. Timing and area results are provided for implementation on a popular FPGA, the Xilinx Virtex 2.

An engineer charged with the design of a modular multiplier for an FPGA project is confronted by a diverse variety of algorithms and architectural alternatives. In addition to the Montgomery and Barrett algorithms, the literature describes Classical modular multipliers (e.g. [3][4]) and Sum of Residues modular multipliers (e.g. [5][6]). A multiplier can be *separated*, so that the product $A \times B$ is computed before being reduced modulo N; or it can be *interleaved*, with reduction occurring digit by digit as partial products are accumulated. Higher radix versions exist for many of the algorithms. Some schemes require a modulus of a specific form, typically $2^n \pm 1$, others permit a general modulus. Finally, as one would expect, micro-architectural optimizations have been published for each algorithm. These may simplify decision logic (e.g. trivial quotient digit selection in [7][8]), permit the use of redundant arithmetic for intermediate results (e.g. [9]), or reschedule the algorithm to move processing steps from the critical path (e.g. [10][11]).

The performance of modular multipliers has been examined at a theoretical level. For example, Montgomery and Classical multipliers are compared in [3], [12] and [13]. Practical results for Montgomery, Classical and Sum of Residues modular multipliers on FPGA are presented in [14]; however this paper does not give timing results for the case of a general modulus N; nor does it consider higher radix or interleaved versions of the algorithms. In this paper, we implement modular multipliers that accept a general modulus N and explore many architectural alternatives to determine their suitability for implementation on FPGA.

A. FPGA Target Technology

To evaluate area and speed of the modular multipliers, the following FPGA technology was used.

Target FPGA: Virtex2 XC2V1000 with a -6 speed grade
1M gates, 5120 slices and 40 embedded 18×18 multipliers
Xilinx 6.1i: XST – Synthesis
ISE – Place and Route
Optimization Goal: Speed

Language: VHDL

Pure delays of the combinatorial circuit were measured excluding those between pads and pins. They were generated from the Post-Place & Route Static Timing Analyzer with a standard place & route effort level.

II. MONTGOMERY MULTIPLICATION

The Montgomery modular multiplication algorithm [1] computes $C = A \times B \times 2^{-n} \bmod N$ where A, B, C and N are all n-bit integers. It has evolved a great deal since its introduction in 1985 [1][16]. In this section we examine the effect of 4 well-known design alternatives: high radix, separated/interleaved structure, trivial quotient digit selection, and quotient digit pipelining.

A. Montgomery Multiplier A

-- High Radix & Separated/Interleaved

Examples of Montgomery multipliers with radices higher than 2 appear in [10] [11] and [15]. A radix-r (r=2^k) version of Montgomery's multiplication algorithm based on [11], which calculates the Montgomery product of A and B, is summarised in the pseudo code in Fig. 1 below, where R is a power of two and greater than N. QDS is a Quotient Digit Selection function.

Fig. 2(a) and 2(b) show the comparison between interleaved and separated multipliers implemented on an FPGA in terms of delays and areas respectively. Various radices are included and n is chosen to be 16.

As can be seen from Fig. 2, the higher the radix $r = 2^k$, the faster the modular multiplier, especially when k is close to the wordlength n. In practice the FPGA technology favors a high-radix approach in which multiplication is performed in few steps using the FPGA's integrated integer multipliers. In the extreme case, where $k = n$, there is no difference between the separated and interleaved approaches.

```
C = 0
For i = 0 to n-1
        C = C + a_i B
        q = QDS(C)
        C = (C + qN)/r
next i
If C >= N Then C = C – N
Return C
```

Fig. 1. Pseudo code for radix-r Montgomery Multiplier A: computing Montgomery product $C = ABR^{-1} \bmod N$

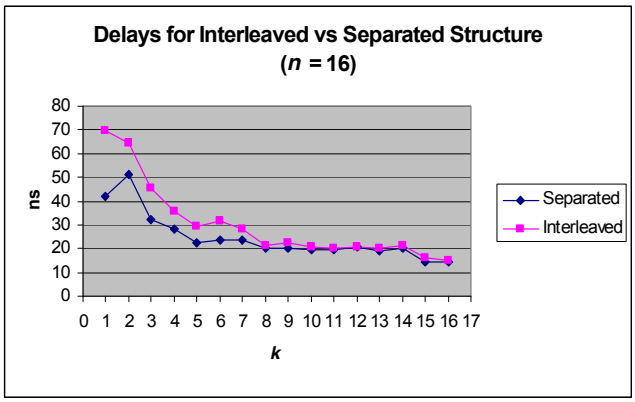

Fig. 2 (a) Delays of Interleaved & Separated Structure at Different Radices

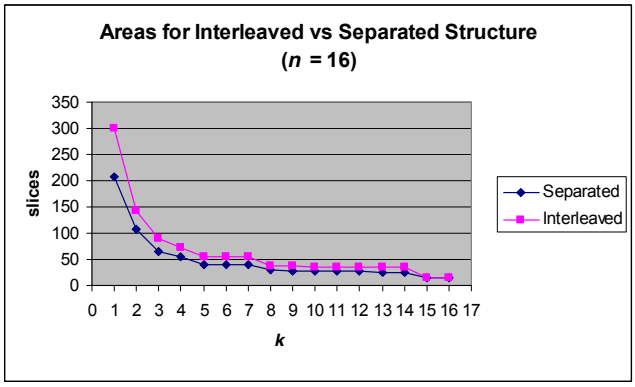

Fig. 2 (b) Area of Interleaved & Separated Structures at Different Radices

B. Montgomery Multiplier B

-- Trivial Quotient Digit Selection

At each iteration the Montgomery reduction algorithm must choose a quotient digit. The Quotient Digit Selection (QDS) logic is on the critical path of a typical Montgomery multiplier. In [11] quotient digit selection is made trivial by assuming the modulus N is always selected such that $N \bmod r = r - 1$. This means that the least significant k bits in the binary representation of N must all be 1. Fig. 3 shows a separated Montgomery Modular Multiplier Algorithm with trivial QDS.

Clearly if k is too close to n, this places an unreasonable restriction on the valid moduli. For our experiments we restrict k such that $k \le \lfloor n/2 \rfloor$. For each wordlength n there will be a value of k that minimizes the delay. Fig. 4 shows the minimum delay observed for n from 12 to 32.

$C_0 = A \times B$
for $i = 0$ to l -1
$C_{i+1} = (C_i + C_i(0)M)/r$
next i
Correct(C_l)

Figure 3. Separated Montgomery Modular Multiplier Algorithm with Trivial QDS

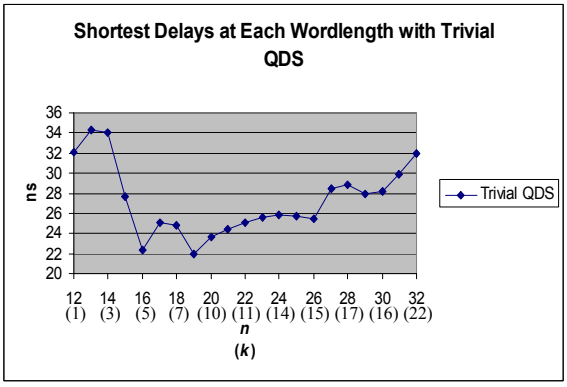

Fig. 4. Delays at n from 12 to 32 using Trivial QDS

C. *Montgomery Multiplier C*
-- *Quotient Pipelining*

The concept of *quotient pipelining* was proposed to move the quotient digit selection from the critical path [10][11]. The algorithm is shown in Fig. 5 with $d \geq 1$, a variable which denotes the level of quotient pipelining.

The $C_i(0)M$ generated in the current iteration is used d iterations later so that C_i does not have to wait for it to generate C_{i+1}.

$C_0 = A \times B$
for $i = 1\text{-}d$ to 0
$C_i(0) = 0$
next i
for $i = 1$ to l
$C_{i+1} = (C_i - C_i(0) + C_{i-d}(0)(M+1)/r^d)/r$
Compute($C_i(0)(M+1)/r^d$)
next i
for $i = l\text{-}d\text{+}1$ to l
$C_{i+1} = C_i + C_i(0)(M+1)/r^{l-i+1}$
next i
Correct(C_l)

Fig. 5. Separated Montgomery Modular Multiplier Algorithm with Trivial QDS and Quotient Pipelining

Quotient pipelining places another restriction on the radix $r = 2^k$. The largest possible k is reduced further by quotient pipelining. It requires that the least significant $k(d+1)$ bits of the modulus N all be 1. We therefore choose k and d to satisfy $k(d+1) \leq \lfloor n/2 \rfloor$. Thus, each wordlength n has a pair of values for k and d that minimizes delay. All the possible values of k and d for n from 12 to 32 were simulated. The shortest delay at each wordlength n is shown in Fig. 6 below.

Fig. 6 also compares the latencies of the Montgomery multiplier before and after using quotient pipelining. The version without quotient pipelining is faster than the quotient pipelining version within the interval $n \in [16, 26]$. In this region the advantage of few, high radix iterations, overwhelms the benefit of more, low radix iterations, despite quotient pipelining and trivial quotient digit selection.

III. BARRETT MULTIPLICATION

The Improved Barrett modular multiplier described in [18] computes $C = A \times B \bmod M = A \times B - \lfloor A \times B \times M^{-1} \rfloor \times M$. It is a separated modular multiplication scheme in which the product $A \times B$ is modulo reduced using multiplication by a pre-computed inverse of the modulus. Specifically, the precomputed inverse:

$$\lfloor A \times B \times M^{-1} \rfloor = \lfloor \lfloor A \times B / 2^{n-2} \rfloor \times \lfloor 2^{2n+3} / M \rfloor / 2^{n+5} \rfloor$$

will have an error of at most 1. Fig. 7 shows the algorithm.

Delay and area results for the Improved Barrett algorithm for $12 \leq n \leq 24$ are shown in Fig. 8. In this implementation the inverse of the modulus M is pre-computed and *hard-coded* into the modular multiplier.

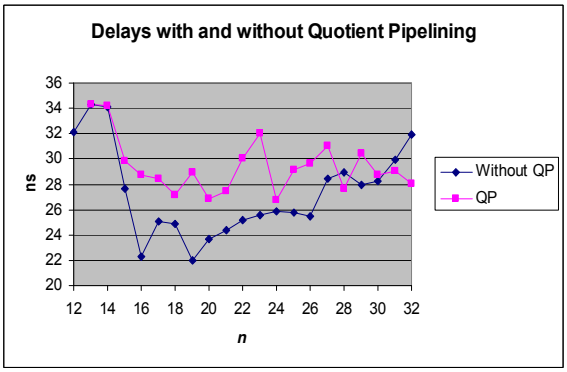

Fig. 6. Comparison of shortest delays with & without Quotient Pipelining at n from 13 to 32. Trivial QDS is used in both cases.

$U = A \times B$

Q = Right shift (U, $n - 2$)

$Q = Q \times K$ // precomputed $K = \lfloor 2^{2n+3} / M \rfloor$

Q = Right shift (Q, $n + 5$)

$C = U - Q \times M$

Fig. 7. Improved Barrett Modular Multiplication Algorithm

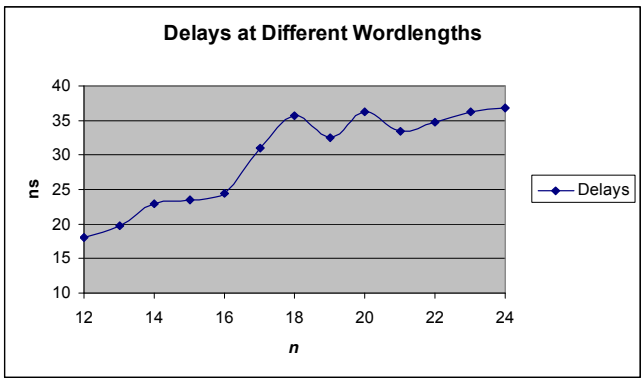

Fig. 8 (a) Delays of the Improved Barrett Modular Multiplier at Different Wordlengths

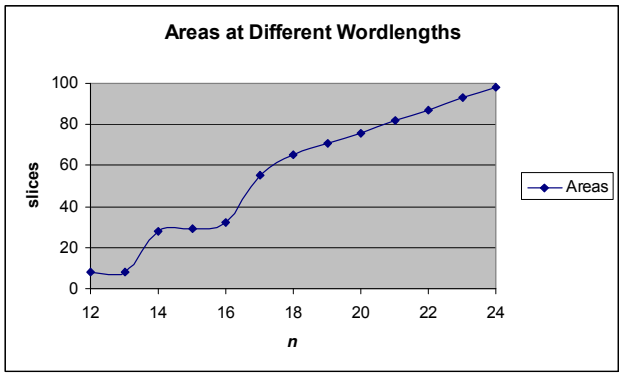

Fig. 8 (b) Areas of the Improved Barrett Modular Multiplier at Different Wordlengths

IV. CONCLUSIONS

Our results are summarized in Fig. 9 (a)(timing) and (b)(areas). The results from [14] are also shown, i.e. the Ma's Group Algorithm [19], Zimmermann's Group Algorithm [20] and the Modified Low-High Algorithm [21]. Note that the same FPGA platform and tools were used in every case. Also note that the three algorithms in [14] are for an interleaved implementation for a modulus of the special form $2^n + 1$.

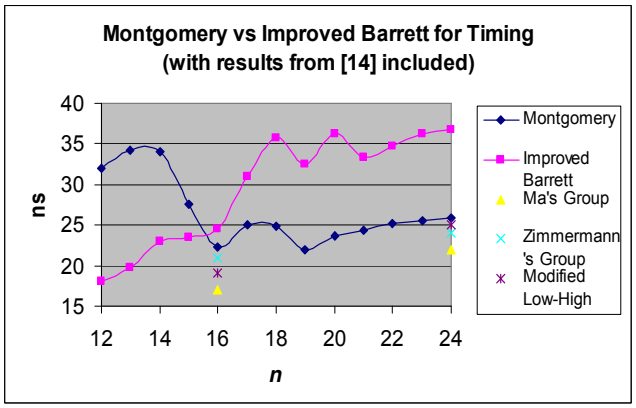

Fig. 9 (a) Comparison of latencies of modular multipliers at different wordlengths

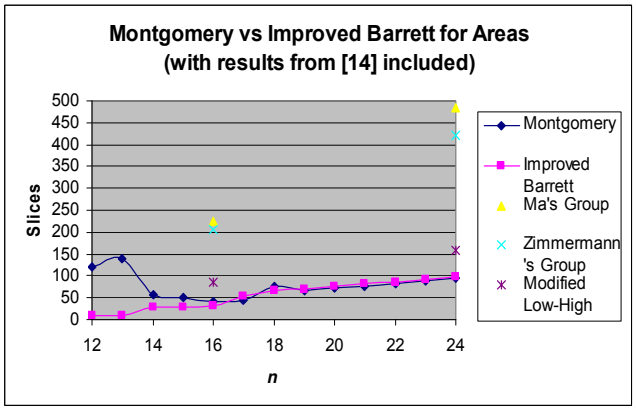

Fig. 9 (b) Comparison of areas of modular multipliers at different wordlengths

For a general modulus, the results show that the best choice between Montgomery and Barrett multiplication depends on the wordlength with Barrett being the better choice for $n < 16$. The presence of hardware integer multipliers on the FPGA influences behaviour strongly in favour of higher-radix and separated approaches.

The algorithms requiring a special form of the modulus from [14] have lower latency than the general modulus algorithms shown here; however the special form algorithms require more area.

ACKNOWLEDGMENT

This research was supported under the Australian Research Council's Discovery/Linkage/Discovery Indigenous Researchers Development/etc funding scheme (project number DP0559582).

REFERENCES

[1] P. L. Montgomery, "Modular Multiplication without Trial Division", *Math. Computation*, Vol. 44, pp. 519-521, 1985.

[2] Paul Barrett. Implementing the Rivest, Shamir and Adleman Public-Key Encryption Algorithm on a Standard Digital Signal Processor. *Lecture Notes in Computer Science:Advances in Cryptology - Crypto 86.* 263:311-323, 1987.

[3] Colin D. Walter, "Montgomery's Multiplication Technique: How to Make It Smaller and Faster", *CHES* 1999: 80-93.

[4] Ernest F. Brickell, "A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography", In *Advances in Cryptology, Proceedings of Crypto 82,* Plenum Press, 1983.

[5] A. Tomlinson, "Bit-Serial Modular Multiplier", *Electronics Letters,* 25(24):1664, November 1989.

[6] C. Y. Chen and C. C. Chang, "A fast modular multiplication algorithm for calculating the product *AB* modulo *N*", *Information Processing Letters*, vol. 72, pp. 77-81, 1999.

[7] Holger Orup and Peter Kornerup, "A High-Radix Hardware Algorithm for Calculating the Exponential M^E Modulo *N*", *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pages 51-57, IEEE Computer Society Press, 1991.

[8] Colin D. Walter, "Faster Multiplication by Operand Scaling", *Lecture Notes in Computer Science: Advances in Cryptology - Crypto 91,* 576:313-323, 1992.

[9] Naofumi Takagi and Shuzo Yajima, "Modular Multiplication Hardware Algorithms with a Redundant Representation and Their Application to RSA Cryptosystem", *IEEE Transactions on Computers,* 41(7):887-890, July 1992.

[10] M. Shand and J. Vuillemin, "Fast Implementations of RSA Cryptography", *Proceedings 11th IEEE Symposium on Computer Arithmetic,* pages 252-259, IEEE Computer Society Press, 1993.

[11] C. D. Walter, "Still faster modular multiplication", *Electronics Letters*, 31(4):263-264, February 1995.

[12] Seong-Min Hong, Sang-Yeop Oh and Hyunsoo Yoon, "New Modular Multiplication Algorithms for Fast Modular Exponentiation", *Lecture Notes in Computer Science:Advances in Cryptology - Eurocrypt 96,* 1070:166-177, 1996.

[13] A. Bosselaers, R. Govaerts and J. Vandewalle, "A Fast and Flexible Software Library for Large Integer Arithmetic", *Proceedings 15th Symposium on Information Theory in the Benelux*, *Louvain-la-Neuve (B)*, pp. 82-89, May 30-31, 1994.

[14] Jean-Luc Beuchat, Laurent Imbert and Arnaud Tisserand, " Comparison of modular multipliers on FPGAs", Proceedings of SPIE - The International Society for Optical Engineering, vol. 5205, 2003, p 490-498.

[15] Colin D. Walter. Systolic Modular Multiplication. *IEEE Transactions on Computers,*42(3):376-378, March 1993.

[16] L. Batina and G. Muurling. Montgomery in practice: How to do it more efficiently in hardware. In B. Preneel, editor, *Proceedings of RSA 2002 Cryptographers' Track*, number 2271 in Lecture Notes in Computer Science, pages 40–52, San Jose, USA, February 18-22 2002. Springer-Verlag.

[17] Y. Kong, "Technical Report on Montgomery Modular Multiplier", CHiPTec Internal Reports, The University of Adelaide, August 2005.

[18] Jean-François Dhem*, Design of an efficient public-key cryptographic library for RISCbased smart cards*. PhD Thesis, Université Catholique de Louvain, May 1998.

[19] Y. Ma, "A simplified architecture for modulo (2^n+1) multiplication," *IEEE Transactions on Computers* 47, pp.333-337, Mar. 1998.

[20] R. Zimmermann, "Efficient VLSI implementation of modulo $(2^n\pm1)$ addition and multiplication," *Proceedings 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia),* Koren and Kornerup, eds., pp.158-167, IEEE Computer Society Press, 1999.

[21] X. Lai, *On the Design and Security of Block Ciphers*, ETH Series in Information Processing, Hartung-Gorre Verlag Konstanz, 1992.