Parallelized Radix-4 Scalable Montgomery Multipliers

Nathaniel Pinckney and David Money Harris
Harvey Mudd College
301 Platt Blvd.
Claremont, CA 91711
{npinckney, David_Harris}@hmc.edu

ABSTRACT

This paper describes a parallelized radix-4 scalable Montgomery multiplier implementation. The design does not require hardware multipliers, and uses parallelized multiplication to shorten the critical path. By left-shifting the sources rather than right-shifting the result, the latency between processing elements is shortened from two cycles to nearly one. The new design can perform 1024-bit modular exponentiation in 8.7 ms and 256-bit exponentiation in 0.36 ms using 5916 Virtex2 4-input lookup tables. This is comparable to radix-2 for long multiplies and nearly twice as fast for short ones.

Categories and Subject Descriptors

B.2.4 [**Arithmetic and Logic Structures**]: High-Speed Arithmetic – *algorithms*, *cost/performance*.

General Terms

Algorithms, Design, Performance.

Keywords

Cryptography, RSA, Montgomery Multiplication.

1. INTRODUCTION

Public key encryption schemes, including RSA, use modular exponentiation of large numbers to encrypt data. This is secure because factoring large numbers is computationally intensive and becomes intractable for very large numbers. But, modular exponentiation of large numbers is slow because of repeated modular multiplications with division steps to calculate the remainder. Montgomery multipliers [1] are useful because they will perform modular multiplication of Montgomery residues without the need of a division step. Hence, they can dramatically increase the speed of encryption systems.

Older Montgomery multipliers are hard-wired to support a particular operand length, n. Scalable Montgomery multipliers reuse w-bit processing elements (PEs) many times to handle the entire n-bit operands, making them suitable to arbitrary-length operands [2]. Previous scalable Montgomery multiplier designs include radix-2 [3, 2], radix-4 [4], radix-8 [5], radix-16 [6], and very high radix [7, 8]. A scalable radix-2^v design processes v bits of the multiplier and w bits of multiplicand per step. The scalable very high radix designs commonly use dedicated $w \times v$ hardware multipliers. These multipliers are efficient on FPGAs containing high-speed multipliers, but may be undesirable for application-specific integrated circuits.

Conventional scalable Montgomery multipliers right-shift the result after each PE. This leads to two-cycle latency between PEs. By left-shifting the operands rather than right-shifting the result, the latency can be reduced to nearly one cycle at the expense of a small increase in the number of iterations through the PEs [3].

The critical path through a PE can be shortened by reordering the steps of the Montgomery multiplication algorithm, which parallelizes multiplications within the PE [9, 7, 6].

This paper describes a novel radix-4 Montgomery multiplier design that left-shifts operands and parallelizes multiplications within the PE. For short operands the multiplier is nearly twice as fast as radix-2 designs and for long operands it is comparable.

2. MONTGOMERY MULTIPLICATION

Montgomery multiplication is defined as

$Z = (XYR^{-1}) \bmod M$

where

X:	n-bit multiplier
Y:	n-bit multiplicand
M:	n-bit odd modulus, typically prime
R:	2^n
R^{-1}:	modular multiplicative inverse of R $(RR^{-1}) \bmod M = 1$

The steps of Montgomery multiplication are shown in Figure 1. Because $R = 2^n$, dividing by R is equivalent to shifting right by n bits. Q has the property that the lower n bits of $[Z + Q \times M]$ are 0. Hence, no information is lost during the reduction step.

Multiply:	$Z = X \times Y$
Reduce:	$Q = Z \times M' \bmod R$
	$Z = [Z + Q \times M] / R$
Normalize:	If $Z \geq M$ then $Z = Z - M$

Figure 1. Montgomery multiplication algorithm

The algorithm involves three dependent multiplications. Orup showed that it can be sped up by reordering steps and doing a precomputation, to eliminate one of the multiplications and to allow the other two to occur in parallel [9].

Note that we can also skip the normalization step for successive Montgomery multiplications because if $R > 4M$ and $X, Y < 2M$ then $Z < 2M$ [9, 10, 11, 12]. To do this we increased the size of the operands to $n_1 = n + 1$ bits and let $R = 2^{n_2}$ where $n_2 = n + 2$.

2.1 Parallelized Radix-4 Scalable Design

The parallelized radix-4 scalable algorithm is a hybrid of two previous Montgomery multiplier designs: the improved unified scalable radix-2 design [3] and the parallelized very high radix scalable design [7]. Figure 2 shows the parallelized radix-4 scalable algorithm derived by [9, 7]. Parallel radix 2^v algorithms require extending the operands by another v bits, so $n_1 = n + 3$ for radix 4. R also increases by 2^v. The variables are defined below.

n_1:	$n + 3$
n_2:	$n + 4$
M:	n-bit odd modulus
M':	n_2-bit integer satisfying $(-MM') \bmod 2^{n_2} = 1$
$\hat{M}$:	n-bit integer $((M' \bmod 2^2) \times M + 1)/2^2$
Y:	n_1 bit multiplicand
X:	n_1 bit multiplier
C:	3-bit carry
w:	scalable inner word length
f:	outer loop length $\left\lceil \frac{n_2}{2} \right\rceil$
e:	inner loop length $\left\lceil \frac{n_1}{w} \right\rceil$

```
$Z = 0$
for i = 0 to $f - 1$
    $Q^i = Z^0 \bmod 2^2$
    $C = 0$
    for $j = 0$ to $e - 1$
        $(C, Z^j) = (Z^{j+1}_{1:0}, Z^j_{w-1:2}) + C + Q^i \times \hat{M}^j + X^i \times Y^j$
```

Figure 2. Parallelized radix-4 scalable Montgomery algorithm

The precomputed $\hat{M}$ is used so that no multiplication is needed to calculate Q. The algorithm is scalable because it iterates over words of the operands using fixed-sized PEs. The superscripts denote 2-bit words for X and w-bit words for Y, Z, and $\hat{M}$. There are $e = \left\lceil \frac{n_1}{w} \right\rceil$ w-bit words of Y, $\hat{M}$, and Z, and $f = \left\lceil \frac{n_2}{2} \right\rceil$ 2-bit words of X in a radix-4 design with w-bit PEs.

3. HARDWARE IMPLEMENTATION

As Tenca proposed [2], the Montgomery multiplier is built from a systolic array of p processing elements (PEs), as shown in Figure 3. The architecture includes memories for X, Y, and $\hat{M}$, a FIFO to store partial words of Z and Q, and a sequence controller. The memory also holds precomputed $3\hat{M}$ and $3Y$ values for multiplications within the PEs. A FIFO holds results of the last PE until the first PE has completed processing the current operands. The FIFO has a latency of b (typically 1) cycles. Bold lines in the figure indicate variables in carry-save redundant form.

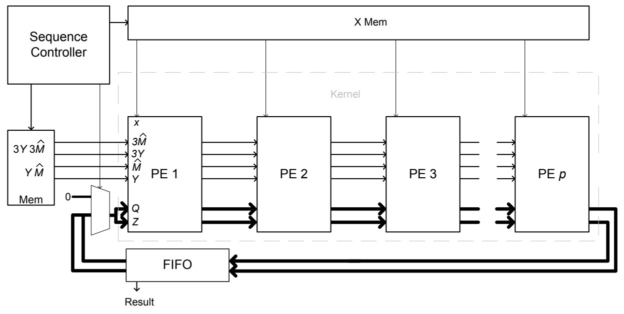

Figure 3. Scalable Montgomery multiplier architecture

3.1 Processing Elements

The parallelized radix-4 processing element design is shown in Figure 4. Each PE receives a different 2-bit word of X, so each PE is assigned a different iteration of the outer loop of the radix-4 algorithm. For a kernel with p PEs, $k = f/p$ pipeline cycles are needed to process all of X. The PEs also receive w bits of Y, $\hat{M}$, Z in each clock cycle. Hence each PE requires e cycles to process all the iterations of the inner loop.

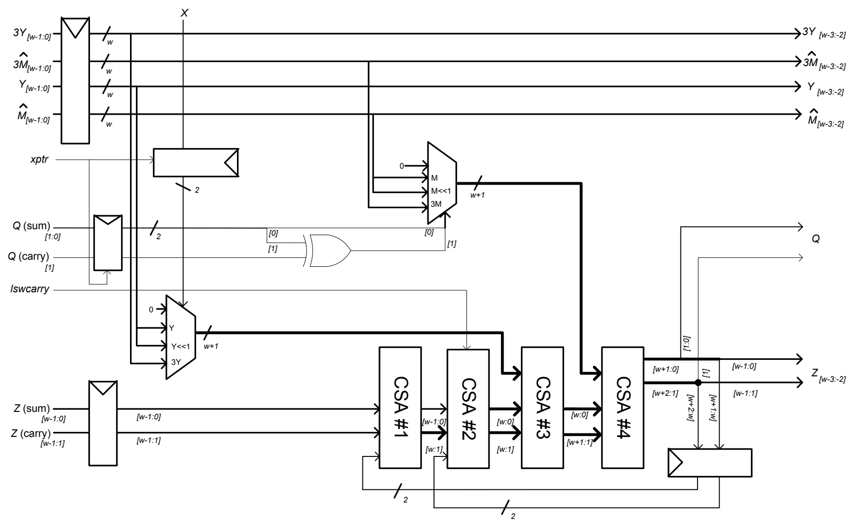

Figure 4. Parallelized Radix-4 scalable Montgomery multiplier processing element

In one pipeline cycle, $2p$ bits of X are processed. Unlike a previous design [7], we ensure that the final result is always taken from the last PE in the kernel to simplify the hardware design. For this to be true, $n_2 = 2pk \geq n + 4$, where k is an integer number of pipeline cycles.

Each PE contains two multiplexers, four 3:2 CSAs, datapath registers (with control signals), and a feedback register for the carry between iterations of the inner loop. Z and Q are represented in carry-save redundant form for speed. Because the CSAs do not have identical length operands, they are optimized into combinations of half adders and full adders to reduce the amount of hardware. Recall that X^i and Q^i can range from 0 to 3 for radix-4. It is trivial to compute $X^i \times Y^j$ or $Q^i \times \widehat{M}^j$ when either multiplier is 2, because Y^j and $\widehat{M}^j$ shift left by 1 bit. Likewise, when the multiplier is 0 or 1 the product is also trivial. When the multiplier is 3, computing the product in real-time would be costly. Instead of including multipliers in the PEs, precomputed $3Y$ and $3\widehat{M}$ are stored in the memory and bussed to each PE, where multiplexers are used to select the product. The drawback of this is extra registers are added to accommodate $3Y^j$ and $3\widehat{M}^j$ in the PE. Since Q is stored in redundant form, it must first be converted to non-redundant form, using an XOR, to select the appropriate multiple of $\widehat{M}$. The multiplexer select lines drive a fanout of $w + 1$, so they must be buffered for adequate drive.

The PE shifts Y^j and $\widehat{M}^j$ left by two bits instead of Z^j right by two after each step. This reduces cycle latency of the PE from two cycles to a single cycle by removing the dependence on the lower bits of the next Z^{j+1} word [3]. As with previous designs, the PE is pipelined for single-cycle throughput. Every $w/2$ steps the lowest word of Z is discarded because it is not in the final result. To simplify implementation, a word is discarded between pipeline cycles. Hence, our design requires that $2p$ must be divisible by w.

So that X^i and Q^i are constant for an entire pipeline cycle, *xptr* is asserted at the start of a pipeline cycle, to enable the X^i and Q^i registers. A shift register, outside of the PE, sequentially asserts PE *xptrs* as words transverse through the kernel.

For conventional Montgomery multiplications using M instead of $\widehat{M}$, the discarded word is all zeros. However, in the parallel version, the word is usually non-zero. Since the result is stored in redundant form, conversion to non-redundant form could produce a carry-out. Logic in the FIFO calculates a carry from the discarded word and outputs to the first PE's *lswcarry*, which is added to the first word of the pipeline cycle. Instead of propagating a carry between PEs through *lswcarry*, the discarded word is processed an extra cycle so that the CSAs can propagate a carry within the PE.

3.2 Latencies

A hardware pipeline diagram of the radix-4 design is shown in Figure 5. The diagram assumes the minimum FIFO cycle latency $b = 1$. Each PE completes a pipeline cycle in e cycles plus an additional cycle to handle overflow because Z is not shifted right after each step.

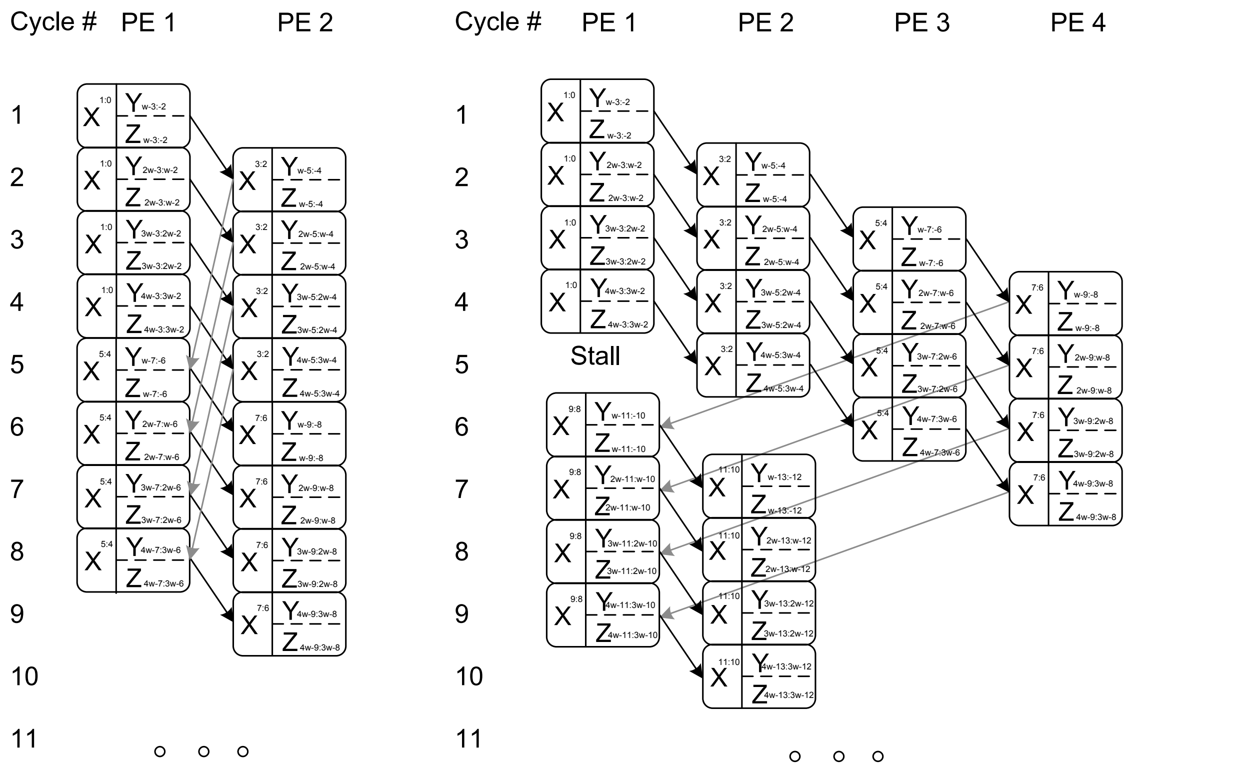

Figure 5. Hardware pipeline diagram for e = 3. p = 2 (Case I, left) and p = 4 (Case II, right)

There are two cases for the time between pipeline cycles. Case I occurs when the PEs are used continuously and is $e + 1$ cycles per pipeline cycle. Case II occurs when the first PE must wait for the result from the last PE. For the first word to transverse through all of the PEs, lp cycles are needed, where l is the cycle latency between PEs (1 in this design). An additional vp/w are needed to handle discarded words of Z as Y and $\widehat{M}$ are shifted. Lastly, an extra b cycles are needed due to latency through the FIFO. Therefore, Case II is $lp + vp/w + b$ cycles per pipeline cycle. The figure is simplified to not show vp/w extra cycles for Case II.

Because k pipeline cycles are needed to process all bits of X, $k(e + 1)$ total cycles are needed in Case I for Montgomery multiplication. Let T_c denote the clock cycle period. The total time for Montgomery multiplication is then

$$T_1 = k(e+1)T_c \quad \text{for } e+1 \geq lp + vp/w + b$$

In Case II, $k(lp + vp/w + b)$ total cycles are needed for Montgomery multiplication. Hence, the total Montgomery multiplication time is

$$T_2 = k(lp + vp/w + b)T_c \quad \text{for } e+1 < lp + vp/w + b$$

Let $m = vwp$ be a metric for the amount of hardware in the multiplier. We can rewrite the above delays in terms of the design parameters n, w, v, p, l, and m, and the low order terms can be dropped, so that the approximate number of cycles is

$$d_1 = \frac{n^2}{m} \qquad \text{for } n \geq lwp$$
$$d_2 = \frac{nl}{v} \qquad \text{for } n < lwp$$

Hence, for Case I the time required for Montgomery multiplication decreases linearly with amount of hardware in the multiplier. For Case II, where operand lengths are short compared to the hardware available, the time does not change with the amount of hardware.

4. RESULTS

The processing elements were coded in Verilog and simulated in ModelSim. Verilog for the parallelized radix-4 design, along with previous Montgomery multiplier designs, has been synthesized in Synplify Pro onto the Xilinx XC2V2000-6 Virtex II FPGA with "Sequential Optimizations" disabled to prevent flip-flops from being optimized into shift registers. A comparison of the parallelized scalable radix-4 design with other Montgomery multiplier designs is shown in Table 1. The critical path for the radix-4 design is a multiplexer, a buffer, two CSAs, and a register, which limits the speed to 266 Mhz for w = 16.

Table 1. Comparison of FPGA resource usage and clock speed

Architecture	Reference	w	v	4-input LUTs / PE	Registers / PE	16 x 16 Mults / PE	Critical Path	Clock Speed (Mhz)
Parallel radix-4 scalable	This work	4	2	55	46	0	2CSA + BUF + MUX + REG	271
		8	2	106	70	0		266
		16	2	187	121	0		266
Parallel radix-2^{16} scalable	[7]	16	16	130	147	2	MUL + CSA + CPA + REG	106
Radix-2^{16} Scalable	[8]	16	16	146	247	2	MUL + CPA + MUX + REG	106
Parallel radix-2 scalable	[13]	16	1	94	72	0	AND + 2CSA + BUF + REG	403
Improved radix-2 scalable	[3]	16	1	95	72	0	2AND + 2CSA + BUF + MUX + REG	285
Tenca-Koç radix-2 scalable	[2]	16	1	95	72	0	2AND + 2CSA + BUF + MUX + REG	285

A comparison of hardware usage and exponentiation time for the parallelized radix-4 design with others is shown in Table 2. The data includes the hardware in the PEs and controller, but not the RAM bits or logic in the memories and FIFO. The modular exponentiation time was calculated by multiplying the time of a single Montgomery multiply with the number of multiplies, $2n + 2$, for a modular exponentiation.

The 32 PE parallelized scalable radix-4 design has the same number of full adder bits $m = vwp$ as a 64 PE radix-2 design. For $w = 16$, the radix-4 design includes 6% fewer LUTs and 20% fewer REGs than the left-shifting scalable radix-2 design of [3]. It performs 256-bit modular exponentiation in 42% less time, but 1024-bit modular exponentiation in 4% more time. The short exponentiation ($n < lwp$) is part of Case II, in which the cycle count scales inversely with v, so we would expect about a 2x speedup. The long exponentiation ($n \geq lwp$) is part of Case I, in which the cycle count scales inversely with m, so we would expect comparable cycle counts. The cycle time is 7% slower for radix 4, so the total time is slightly faster than cycle counts would directly predict. The parallelized radix-4 design also come at the expense of precomputing and storing $3Y$ and $3\widehat{M}$.

Compared to a conventional right-shifting scalable radix 2 design of [2] with the same hardware count, m, the parallel design has comparable cycle counts for long operands but one fourth the cycle count for short operands, because of the higher radix and the shorter latency between PEs.

Table 2. Comparison of Modular Exponentiation Times

Description	Ref	Technology	Freq (MHz)	*w*	*v* *(radix -2^v)*	*p*	LUTs	REGs	16 x 16 MULTs	n	T (ms)
Parallel radix-4 scalable	This work	Xilinx XC2V2000-06	266	16	2	16	2996	1955	0	256	0.33
										1024	17
						32	5916	3863	0	256	0.36
										1024	8.7
						64	11684	7642	0	256	0.42
										1024	5.1
Parallel radix-2^{16} scalable	[7]	Xilinx XC2V2000-06	106	16	16	4	739	611	8	256	0.43
										1024	22
						16	2657	2375	32	256	0.32
										1024	6.4
Radix-2^{16} scalable	[8]	Xilinx XC2V2000-06	106	16	16	4	686	980	8	256	0.41
										1024	21
						16	2438	3944	32	256	0.63
										1024	6.3
Radix-4 scalable	[4]	0.5 μm CMOS	181	8	2	16	10 Kgates		0	256	0.95
										1024	53
Radix-8 scalable	[5]	0.5 μm CMOS	88	8	3	16	16 Kgates		0	256	1.3
										1024	66
Parallel improved radix-16 scalable	[6]	0.25 μm CMOS	180	32	4	32	150 Kgates		0	1024	3.4
Parallel radix-2 scalable	[13]	Xilinx XC2V2000-06	403	16	1	16	1575	1189	0	256	0.41
										1024	21.8
						64	6006	4597	0	256	0.52
										1024	6.3
Improved radix-2 scalable	[3]	Xilinx XC2V2000-06	285	16	1	16	1408	1205	0	256	0.55
										1024	30
						64	6317	4844	0	256	0.62
										1024	8.4
Tenca-Koç radix-2 scalable	[2]	0.5 μm CMOS	192	8	1	40	28 Kgates		0	265	1.6
										1024	37
		Xilinx XC2V2000-06	285	8	1	40	3902	2937	0	256	1.0
										1024	15

5. CONCLUSIONS

This paper describes a parallelized scalable radix-4 Montgomery multiplier design that reduces PE cycle latency and critical path length. The design is suitable for situations where a dedicated hardware multiplier is undesirable or not available. The design performs 2-4 times as fast as previous radix-2 designs for small multiplies, but about 4% slower for large multiplies because the cycle time is somewhat longer. Each radix-4 PE uses 1.97 times as many LUTs as a radix-2 PE, so the total hardware consumption per bit processed per cycle is comparable.

Future radix-4 designs could use Booth encoding instead of precomputed $3Y$ and $3\hat{M}$ values. This might deliver similar performance with fewer registers needed to store precomputed values.

6. ACKNOWLEDGMENTS

The authors thank the Clay-Wolkin Family Foundation fellowship and Intel Circuit Research Lab for funding the research.

7. REFERENCES

[1] P. Montgomery, "Modular multiplication without trial division," *Math. of Computation*, vol. 44, no. 170, pp. 519-521, April 1985.

[2] A. Tenca and Ç. Koç, "A scalable architecture for modular multiplication based on Montgomery's algorithm," *IEEE Trans. Computers*, vol. 52, no. 9, Sept. 2003, pp. 1215-1221.

[3] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, "An improved unified scalable radix-2 Montgomery multiplier," Proc. 17th *IEEE Symp. Computer Arithmetic*, pp. 172-178, 2005.

[4] A. Tenca and L. Tawalbeh, "An efficient and scalable radix-4 modular multiplier design using recoding techniques," *Proc. Asilomar Conf. Signals, Systems, and Computers*, pp. 1445-1450, 2003.

[5] A. Tenca, G. Todorov, and Ç. Koç, "High-radix design of a scalable modular multiplier," *Cryptographic Hardware and Embedded Systems*, Ç. Koç and C. Paar, eds., 2001, Lecture Notes in Computer Science, No. 1717, pp. 189-206, Springer, Berlin, Germany.

[6] Y. Fan, X. Zeng, Y. Yu, G. Wang, and Q. Zhang, "A modified high-radix scalable Montgomery multiplier," *Proc. Intl. Symp. Circuits and Systems*, pp. 3382-3385, 2006.

[7] K. Kelley and D. Harris, “Parallelized very high radix scalable Montgomery multipliers,” *Proc. Asilomar Conf. Signals, Systems, and Computers*, pp. 1196-1200, Nov. 2005.

[8] K. Kelley and D. Harris, “Very high radix scalable Montgomery multipliers,” *Proc. 5th Intl. Workshop on System-on-Chip*, pp. 400-404, July 2005.

[9] H. Orup, “Simplifying quotient determination in high-radix modular multiplication,” *Proc. 12th IEEE Symp. Computer Arithmetic*, pp. 193-199, July 1995.

[10] T. Blum and C. Paar, “High-radix Montgomery multiplication on reconfigurable hardware,” *IEEE Trans. Computers*, vol. 50, no. 7, July 2001, pp. 759-764.

[11] C. Walter, “Montgomery exponentiation needs no final subtractions,” *Electronics Letters*, vol. 35, no. 21, pp. 1831-1832, 14 October 1999.

[12] G. Hachez and J. Quisquater, “Montgomery exponentiation with no final subtractions: improved results,” *Lecture Notes in Computer Science*, C. Koç and C. Paar, eds., vol. 1965, pp. 293-301, 2000.

[13] N. Jiang and D. Harris, “Parallelized Radix-2 Scalable Montgomery Multiplier,” submitted to *IFIP Intl. Conf. on VLSI*, 2007.

David Money Harris is an Associate Professor of Engineering at Harvey Mudd College. He received his Ph.D. in Electrical Engineering from Stanford in 1999, his M.Eng. degree in Electrical Engineering and Computer Science from MIT in 1994, and his S.B. degrees in Mathematics and Electrical Engineering from MIT in 1994. Dr. Harris has worked and consulted at Intel Corporation, Sun Microsystems, and numerous other companies in the area of integrated circuit design. He is the author of four textbooks in the field.

Nathaniel Pinckney is a Clay-Wolkin fellow and senior engineering major at Harvey Mudd College. He plans to pursue a Ph.D after graduation. He has interned at Applied Minds and at a young startup company.