Parallelized Radix-2 Scalable Montgomery Multiplier

Nan Jiang and David Harris
Harvey Mudd College
301 E. Twelfth St. Claremont, CA 91711
{Nan_Jiang, David_Harris}@hmc.edu

***Abstract*- This paper describes the FPGA implementation of a parallelized scalable radix-2 Montgomery multiplier. It improves upon previous designs by rearranging previously sequential calculations to take place in parallel. On a Virtex-II FPGA, this design can perform 1024-bit modular exponentiation in 6.3 ms using 6006 lookup tables, a 17% speed improvement over the previously fastest scalable radix-2 Montgomery multiplier.**

I. Introduction

Modular exponentiation is widely used in modern cryptography algorithms such as RSA and digital signatures. However, the operands of these algorithms usually involve 256 to 2048-bit numbers and the process is time-consuming due to the long divisions necessary in the modulo calculation. Montgomery multiplication transforms this difficult division into a simple bit shift, and is therefore very attractive for hardware implementation.

Since the advent of Montgomery's algorithm in [1], there have been many designs of the Montgomery multiplier that falls into different categories based on their radix. In a simple radix-2 design, an $n \times n$-bit multiplication is performed using n steps. The multiplier kernel contains processing elements (PEs) that acts on one bit of the multiplier and all n bits of the multiplicand. These designs are hardwired to support only one choice of n. A *scalable* radix-2 design described in [2, 3] breaks the n-bit multiplicand into w-bit words. The kernel of the designs contain p PEs organized in a systolic array. Each PE handles one bit of the multiplier and w bits of the multiplicand at a time. The kernel iterates until the entire multiplication completes. These designs are highly flexible; they can be configured to handle any n. The overall advantage of a radix-2 Montgomery multiplier design is its hardware simplicity. The $1 \times n$-bit or $1 \times w$-bit multiplication involved in these designs can be accomplished with an n-bit AND gate or a multiplexer, and the number of registers required to store intermediate values is minimal. However, the hardware efficiency comes at the cost of large number of iterations through the kernel.

The critical path in a standard Montgomery multiplier involves two dependent multiplications steps. Orup showed how to reorder the steps so that the multiplications can take place in parallel [7]. We have successfully applied the technique to very high radix Montgomery multipliers [4, 5, 6] to shorten the critical path. Our goal in this paper is to apply the parallel modification of the Montgomery algorithm to the improved scalable radix-2 multiplier implemented in [3]. By parallelizing the existing radix-2 designs, we achieve a significant performance boost without increasing hardware cost.

II. Background

The basic Montgomery Multiplication algorithm is

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

with the notations

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 satisfying $(RR^{-1}) \bmod M = 1$
M':	n-bit integer satisfying $RR^{-1} - MM' = 1$

Montgomery in [1] showed how to perform this multiplication without dividing by M:

Multiply:	$Z = X \times Y$
Quotient:	$Q = Z \times M' \bmod R$
Result :	$Z = [Z + Q \times M] / R$

The Q term has the property such that it forces the numerator of the Result step to be divisible by R, simplifying the division to a shift.

A. Improved Radix-2 Design

The radix-2 design in [3] will be the basis of this paper's radix-2 implementation. The design follows Tenca-Koç's multiple word radix-2 Montgomery multiplication algorithm from [2] shown in Figure 1.

n:	size of operands
R:	2^n
M:	n-bit odd modulus
R^{-1}:	modular multiplicative inverse of R satisfying $(RR^{-1}) \bmod M = 1$

M': n-bit integer satisfying $(RR^{-1}-MM') = 1$
X : n-bit muliplier
Y: n-bit multiplicand

$Z = 0$
for $i = 0$ to $n-1$
    $Z = Z + X^i \times Y$
    if Z is *odd* then $Z = Z + M$
    $Z = Z/2$

Figure 1: Tenca-Koç's radix-2 Montgomery multiplication algorithm

Each iteration of the i loop is called a kernel cycle, and the clock delay between successive PEs is called a PE cycle. The Tenca-Koç design [2] requires a delay of two clock cycles between PEs, because at the end of a PE the resulting word of Z needs to be right shifted to account of the division by 2. Before this shift can occur, the next word of Z needs to be calculated so that the least significant bit of the next word of Z can be shifted to become the most significant bit of the current word of Z. Our improved design [3] is able to avoid right shifting of Z by left shifting Y and M, as show Figure 2. Each PE no longer has to wait for the next word of Z to be computed before producing the correct Z output. Thus, the latency between PEs drops to only one clock cycle.

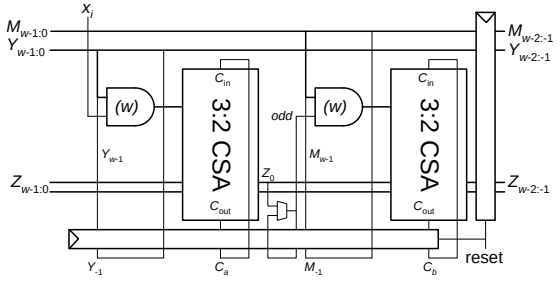

Figure 2: PE diagram for improved radix-2 design from [3]

B. Parallelized Algorithm

In [5], an alternative implementation of the Montgomery algorithm based on [7] is discussed. The parallel algorithm reorganizes the original algorithm to produce a new pre-calculated value $\hat{M}$ that allows the original algorithm's Multiply and Result steps' multiplications to occur simultaneously. After parallelizing, the basic algorithm is shown in Figure 3

$\hat{M}$: $((M' \bmod 2)M + 1)/2$

$Z = 0$
for $i = 0$ to n
    $Q = Z \bmod 2$
    $Z = Z/2 + Q \times \hat{M} + X^i \times Y$

Figure 3: Parallelized Montgomery algorithm

There are two side effects caused by parallelizing the original algorithm. The first is that the result, Z, has increased in size by 1 bit. This is because division by 2 is no longer the final operation of the algorithm. This side effect requires additional calculations to normalize the result back to the expected range. However, normalization can be performed at the end of a modular exponentiation after many iterations of Montgomery multiplication. The second side effect is that this design requires one more kernel cycle than the non-parallel algorithm to compensate for the loop reordering.

III. PARALLELIZED RADIX-2

The Montgomery algorithm implemented for this paper is a hybrid between the improved radix-2 and the parallel algorithm. The basic algorithm is nearly identical to the parallel very high radix algorithm in Figure 3. The features of the improved radix-2 algorithm are introduced by left shifting $\hat{M}$ and Y at each PE. The resulting algorithm takes advantage of both simultaneous multiplication and one-cycle latency between PEs.

The general algorithm can be recast in scalable form by splitting $\hat{M}$ and Y into w-bit words. Each PE has to run multiple times during a kernel cycle to process all bits of $\hat{M}$ and Y. Thus, an inner for loop iterates over the n/w words of $\hat{M}$ and Y. In addition, another iteration of the inner loopis necessary to process the left shifted bits of $\hat{M}$ and Y. Furthermore, as mentioned previously, one side effect of the parallel algorithm is that the result could be larger than expected. Thus the inner loop iterations is increase by one to account for the additional PE iteration required. The resulting scalable algorithm is shown in Figure 4.

w: multiplicand word length
e: $\lceil n/w \rceil + 2$ PE iterations per kernel
C: 1-bit carry digit

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

Figure 4: Scalable parallel Radix-2 algorithm

IV. HARDWARE IMPLEMENTATION

The overall hardware architecture of the parallel radix-2 multiplier is similar to those presented in [3, 4, 5, 6]. Figure 5 provides the overview architecture of a scalable Montgomery multiplier using p PEs. Every PE receives one bit of X and Q, and w bits of $\hat{M}$, Y, and Z on each step. In one kernel cycle, p digits of X are processed against all n bits of $\hat{M}$ and Y'. Hence, $k = \lceil n/p \rceil + 1$ full kernel cycles are necessary to process all the bits of X and satisfy the additional kernel cycle requirement of parallel algorithm. As results emerge from the

last PE of the kernel, they are either stored in a FIFO until the first PE has finished its kernel cycle or bypassed directly to the input of the first PE.

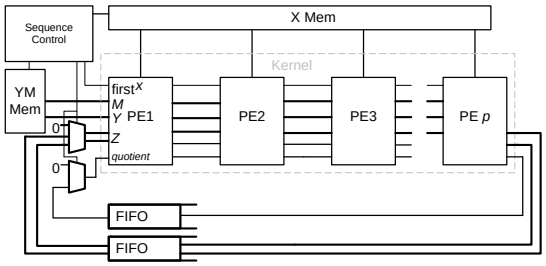

Figure 5: Parallel radix-2 hardware diagram

A. Processing Element

Figure 6 shows a processing element for the parallel radix 2 design. Compared to Figure 2, the two AND-multipliers are placed in parallel and the 2 input multiplexer is eliminated. The left shifting of $\hat{M}$ and Y are achieved using the two delay registers in the design. At each PE, the most significant bit of a word of $\hat{M}$ and Y are delayed to become the least significant bit of the next word of $\hat{M}$ and Y. After $\hat{M}$ and Y have passed through p PEs, they are in effect left shifted by p bits. The PE diagrams also show that the change from improved radix-2 design to parallel radix-2 design eliminates an AND and a multiplexer from the critical path without increasing the hardware cost.

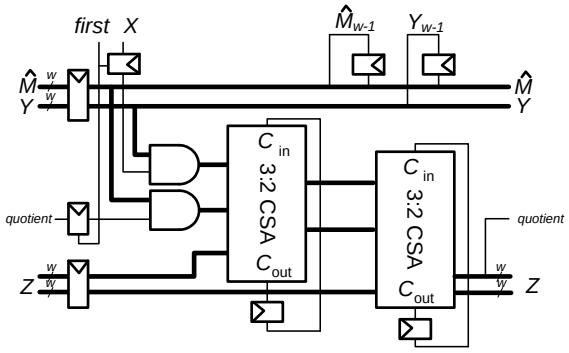

Figure 6: Parallel Radix-2 PE diagram

B. Latency

The latency of the parallelized radix-2 design is similar to that of [3]. One PE cycle consists of only one clock cycle due to the left shifting of $\hat{M}$ and Y. Each PE multiplies one bit of X with w bits of Y *and* $\hat{M}$. When a PE has processed all the bits of $\hat{M}$ and Y, a kernel cycle has completed. It will then wait for a new set of X bits to start the cycle all over again. Modification introduced by the parallel algorithm shows no visible effect on the latency graph in Figure 7. However, e and k are increased by 1.

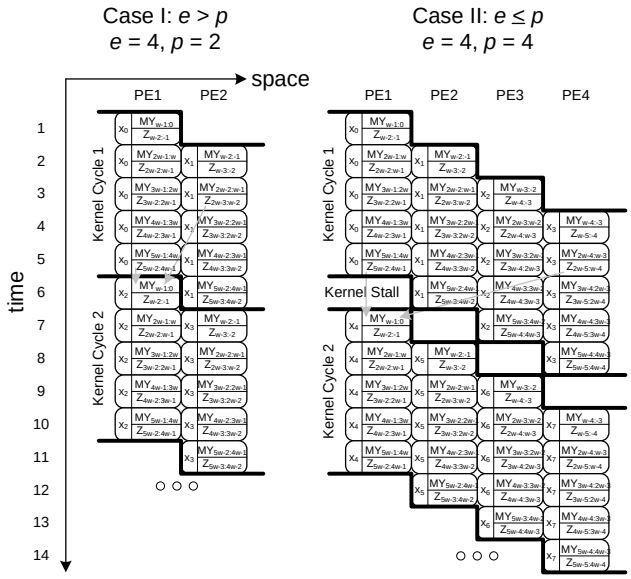

Figure 7: Parallel radix-2 kernel latency

An entire multiplication using p PEs takes k kernel cycles to complete. When the first PE has finished a kernel cycle, it cannot begin the next kernel cycle until the last PE has completed the first word of Z. The latency of a kernel cycle depends on e and p. Case I corresponds to a large number of PE cycles, e, relative to the number of processing elements, p. In this situation, when the first PE has finished its kernel cycle, the first word of Z from the last PE is already waiting in the FIFO, there is no stall between kernel cycles, and the kernel hardware is used with maximal efficiency. Case II corresponds to a large number of processing elements relative to the number of PE cycles. As shown in Figure 7, the first PE must wait until the last PE finishes calculating the first word of Z. Therefore, Case I occurs when $e > p$ and Case II occurs when $e < p+1$.

Case I: The first PE is used continuously e times per kernel cycle for k full kernel cycles. Therefore the total delay is

$$D_I = ke \qquad (2)$$

Case II: Each kernel cycle takes $p+p/w$ clock cycles until the first word of Z is ready, plus 1 to bypass the result back to the first PE. The p/w term is caused by the left shifting of $\hat{M}$ and Y. After passing p PEs, p zeros have been shifted into the least significant bits of $\hat{M}$ and Y which are ignored, causing p/w cycles of delay. Therefore the total delay Case II is

$$D_{II} = k(p+p/w+1) \qquad (3)$$

Rewriting these delays in terms of the design parameters n, w, and p, and assuming integer divisibility, we obtain

TABLE 1: SYNTHESIS PERFORMANCE COMPARISON OF VARIOUS PROCESSING ELEMENTS

Architecture	**Reference**	***w***	**4-input LUTs / PE**	**Registers / PE**	**Critical Path**	**PE Clock Speed (MHz)**
Parallel Scalable Radix 2	This work	16	94	72	AND + 2CSA + REG	403
Improved Scalable Radix 2	[3]	16	97	72	2AND + 2CSA + BUF + MUX + REG	318
Tenca-Koç Scalable Radix 2	[2]	16	97	72	2AND + 2CSA + BUF + MUX + REG	318

$$D_I = \frac{n^2}{pw} + \frac{n}{p} + \frac{n}{w} \quad (4)$$

$$D_{II} = n + p + \frac{n+p}{w} + \frac{n}{p} + 1 \quad (5)$$

V. RESULTS

The parallel radix-2 Montgomery multiplier design described previously was implemented using Verilog. The design was synthesized using Synplify Pro and compared to the synthesis result of previous designs. All synthesis results were produced by targeting the Xilinx Virtext II XC2V2000 speed grade -6 FPGA with "sequential optimizations" disabled [8] to prevent flip-flops from being turned into shift registers. Table 1 shows the synthesis result of a single process element for several radix-2 designs. The result matches our expectation that the parallel algorithm removed an AND and a multiplexer from the PE critical path. As a result, the PE of a parallel radix-2 design achieved a 26% clock speed increase. In addition, because several gates were removed from the parallel radix-2 PE, there is a hardware decrease when compared to previous designs. Thus, the parallel radix-2 design is both faster and small than previous designs.

Table 2 compares overall system performance of the new parallel design to the other scalable radix-2 designs. For the kernel synthesis, the frequency of the kernels is significantly lower than the PE synthesis show in Tables 1. This difference is caused by the interconnect delay estimated by Synplify Pro which can be minimized in a real implementation of the Montgomery multipliers through datapath floorplanning.

From Table 2, we see that the parallel radix-2 design is the fastest radix-2 multiplier. In the 1024-bit multiplication using 16 PEs, it can perform a multiplication 17% faster than the improved radix-2 design and significantly faster than the improved radix-2 design. In addition, the hardware requirement of the parallel radix-2 multiplier is approximately the same as other designs, justifying our claim that this design provides an performance increase without additional hardware costs.

It is interesting to note that as the number of PEs in the kernel increase, the performance benefit of the parallel radix-2 design decrease. This effect is caused by the fact parallel algorithm requires an addition kernel cycle than traditional radix-2 Montgomery multipliers. Thus, as the number of PE increases, this performance overhead for parallel radix-2 is increased as well. It is possible to increase the number of PEs so much that the parallel radix-2 design would actually become slower than the traditional designs despite of running at higher clock frequencies.

VI. CONCLUSION

In this paper, we have demonstrated a novel approach to radix-2 scalable Montgomery multipliers by reordering the steps to perform multiplications in parallel. The design is both faster and smaller than previous radix-2 Montgomery multipliers. It provides a significant cycle time improvement at the cost of a small increase in cycle count. In simulation, the parallel radix-2 design was able to provide a 17% speed increase over the previous designs for a 1024-bit multiplication using 16 PEs.

TABLE 2: SYNTHESIS PERFORMANCE COMPARISON OF VARIOUS MONTGOMERY MULTIPLIERS

Description	**Ref**	**Tech**	***w***	***v***	***p***	**LUTs**	**REGs**	**16×16 MULT**	***N***	T_{mult} **(ms)**
Parallel Scalable radix 2	This work	Xilinx XC2V2000-06	16	1	16	1575	1189	0	256	0.41
									1024	21.8
					64	6006	4597	0	256	0.52
									1024	6.3
Improved scalable radix 2	[3]	Xilinx XC2V2000-06	16	1	16	1564	1202	0	256	0.49
									1024	27.2
					64	5932	4705	0	256	0.56
									1024	7.6
Tenca-Koç Scalable radix 2	[2]	0.5 μm CMOS	8	1	40	28 kgates		0	256	1.6
									1024	37
		Xilinx XC2V2000-06	8	1	40	3902	2937	0	256	1.0
									1024	15

ACKNOWLEDGEMENT

The authors would like to thank the Clay-Wolkin Family Foundation fellowship as well as Intel Circuit Research Lab for funding this research project.

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, pp. 1215-1221, Sept. 2003.

[3] D. Harris *et al*., "An improved unified scalable radix-2 Montgomery multiplier", *IEEE Symp. Computer Arithmetic*, pp. 172-178, 2005.

[4] N. Jiang and D. Harris, "Quotient piplelined very high radix scalable Montgomery multipliers", *Proc. Asilomar Conf. Signal, Systems , and Computers*

[5] K. Kelly and D. Harris, "Parallelized very high radix scalable Montgomery multipliers," *Proc. Asilomar Conf. Signals, Systems, and Computers*, pp. 1196-1200, 2005.

[6] K. Kelley and D. Harris, "Very high radix scalable Montgomery multipliers", *IEEE IWSOC Conference*, pp. 400-404, July 2005.

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

[8] Xilinx, Virtex-II Pro and Virtex-II Pro X Platform FPGAs Datasheet, June 30, 2004, www.xilinx.com