Accelerating Montgomery Modulo Multiplication for Redundant Radix-64k Number System on the FPGA using Dual-Port Block RAMs

Koji Shigemoto, Kensuke Kawakami, Koji Nakano
Department of Information Engineering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, JAPAN

Abstract

The main contribution of this paper is to present hardware algorithms for redundant radix-2^r number system in the FPGA to accelerate Montgomery modulo multiplication with many bits, which have applications in security systems such as RSA encryption and decryption. Quite surprisingly, our hardware algorithm for Montgomery modulo multiplication of two dr-bit numbers can be completed in only $d+1$ clock cycles. Since most FPGAs have 18-bit multipliers and 18k-bit block RAMs, it makes sense to let $r = 16$. Our hardware algorithm for Montgomery modulo multiplication for 256-bit numbers runs only 17 clock cycles using redundant radix-$64k$ (i.e. radix-2^{16}) number system. The experimental results for Xilinx Virtex-II Pro Family FPGA XC2VP100-6 show that the clock frequency of our circuit is independent of d. Further, the hardware algorithm for 1024-bit Montgomery modulo multiplication using the redundant number system is 3 times faster than that using the conventional number system. Also, for 256-bit Montgomery modulo multiplication, our hardware algorithm runs in 0.322μs, while a previously known implementation runs in 1.22μs although our implementation uses less than a half slices.

1 Introduction

An FPGA (Field Programmable Gate Array) is a programmable VLSI in which a hardware designed by users can be embedded instantly. Typical FPGAs consist of an array of programmable logic blocks (slices), memory blocks, and programmable interconnect between them. The logic block contains four-input logic functions implemented by a look up table and/or several registers. Using four-input logic functions, registers, and their interconnects, any combinational circuits and sequential logic can be implemented. The memory block is a dual-port RAM which can perform read and/or written operations for a word of data to two distinct or same addresses in the same time. Usually, the dual-port RAM supports synchronous read and synchronous write. The read and/or write operations are performed at the rising edge of clock if read and/or write enable inputs are high. The dual-port RAM outputs data in a specified address after the rising edge. Similarly, data is written to a specified address at the rising edge of clock if write enable is high. Thus, a clock cycle necessary to perform read/data operations. Using design tools provided by FPGA vendors or third party companies, a hardware logic designed by users using hardware description languages can be embedded in FPGAs. It has been shown that a lot of computation can be accelerated using a circuit implemented in FPGAs [3, 4, 7, 11, 12].

It is well known that the addition of two n-bit numbers can be done using a ripple carry adder with the cascade of n full adders [5]. The ripple carry adder has a carry chain through all the n full adders. Thus, the delay time to complete the addition is proportional to n. The carry look-ahead adder [5, 13] which computes the carry bits using the prefix computation can reduce the depth of the circuit. Although the delay time is $O(\log n)$, its constant factor is large and the circuit is much more complicated than the ripple carry adder. Hence, it is not often to use the carry look-ahead adder for actual implementations. On the other hand, redundant number systems can be used to accelerate addition. Using redundant number systems, we can remove long carry chains in the addition. The readers should refer to [13] (Chapter 3) for comprehensive survey of redundant number systems.

The main contribution of this paper is to present hardware algorithms for redundant radix-2^r number system in the FPGA to speed Montgomery modulo multiplication [10], which have applications in security systems such as RSA encryption and decryption system [14]. Montgomery modulo multiplication is used to speed the modulo multiplication $X \cdot Y \cdot 2^{-R} \bmod M$ for R-bit numbers X, Y, and M. The idea of Montgomery modulo multiplication is not to use direct modulo computation, which is very costly in terms of the computing time and hardware resources. By iterative computation of Montgomery modulo

978-0-7695-3492-3/08 $25.00 © 2008 IEEE
DOI 10.1109/EUC.2008.30

multiplication, the modulo exponentiation $P^E \bmod M$ can be computed, which is a key operation for RSA encryption and decryption [2].

This paper shows implementations of redundant radix-2^r number system in the FPGA for arithmetic operations and then presents hardware algorithms for Montgomery modulo multiplication. The key feature of our implementation for Montgomery modulo multiplication is to use

- redundant radix-2^r number system for interim results,
- dual-port block RAMs to compute $k \cdot M$ such that $(X \cdot Y + k \cdot M) \bmod 2^r = 0$, and
- 18-bit multipliers in an effective manner.

Since each digit of redundant radix-2^r number system has $r+2$ bits and most FPGAs has 18-bit multiplies as building blocks, it makes sense to let $r = 16$.

Quite surprisingly, our hardware algorithms for Montgomery modulo multiplication runs in $d+1$ clock cycles for d-digit redundant radix-2^r number system which takes integers up to $2^{dr}-1$. For example, our hardware algorithm for 1024-bit Montgomery modulo multiplication runs for $1024/16+1 = 65$ clock cycles. From the experimental results for the Xilinx Virtex II Pro Family FPGA XC2VP100-6. our implementation for 1024-bit Montgomery modulo multiplication runs in 1.23μs. Further, the speed up factors of our hardware algorithm using the redundant number system over those using the conventional number system are 3 for 1024-bit Montgomery modulo multiplication.

Several hardware implementations have been presented for Montgomery modulo multiplication [2, 6, 8, 9]. For example, it was shown in [8] that 256-bit Montgomery modulo multiplication can be done in 93 clock cycles of 76.17 MHz (=1.22μs) using 64 18-bit multipliers and 4663 slices on the Xilinx Virtex II Pro Family FPGA XC2VP125-7. In our implementation, it can be done in 17 clock cycles of 52.86MHz (=0.322μs) using 2054 slices and 16 18-bit multipliers and 8 18k-bit block RAMs. Table 1 summarizes the comparison of the performance for 256-bit Montgomery modulo multiplication. Our implementation is more than five times faster although it uses lesser hardware resources on a lesser grade FPGA. Also, the implementation presented in [1] uses 16 times more slices than ours.

2 Non-Redundant and Redundant Radix-2^r Numbers

In this paper, we use the following notation to represent the consecutive bits in a number. For a number X, let $X[i,j]$ $(i \geq j)$ be consecutive bits from i-th to j-th bits, where the least significant bits is 0-th bit. For example, $X[6,2] = 11100$ for $X = 11110000$.

Before defining redundant radix-2^r number, we start with a non-redundant radix-2^r number. *A d-digit non-redundant radix-2^r number* is a sequence X of d r-bit numbers $(X_{d-1}, X_{d-2}, \ldots, X_0)$. The value of X is $\sum_{i=0}^{d-1} X_i \cdot 2^{ir}$ and, it takes an integer up to $\sum_{i=0}^{d-1}(2^r-1)\cdot 2^{ir} = 2^{dr}-1$. Hence, it is also a conventional dr-bit binary number. Also, for any integer X, its d-digit non-redundant radix-2^r number $(X_{d-1}, X_{d-2}, \ldots, X_0)$ is unique.

A d-digit redundant radix-2^r number is a sequence X of d $(r+2)$-bit numbers $(X_{d-1}, X_{d-2}, \ldots, X_0)$. The value of X is $\sum_{i=0}^{d-1} X_i \cdot 2^{ir}$. We call, for each X_i with $r+2$ bits, $X_i[r-1,0]$ and $X_i[r+1,r]$, *principal bits* and *redundant bits*, respectively. For example, $X = (\underline{00}0101, \underline{01}0011, \underline{11}1111, \underline{10}1111)$ is a 4-digit redundant radix-2^4 number, where underlined binary numbers are redundant bits. The value of X can computed as follows:

$$\begin{array}{cccccc} & \underline{00} & \underline{01} & \underline{11} & \underline{10} & \\ + & & 0101 & 0011 & 1111 & 1111 \\ \hline & 00 & 0110 & 0111 & 0001 & 1111 \end{array}$$

Clearly, for any integer X, its d-digit redundant number may not be unique. For example, the value of $Y = (\underline{00}0110, \underline{00}0111, \underline{00}0001, \underline{00}1111)$ is equal to that of X although they have different numbers in each corresponding digit. Since the all the redundant bits of this redundant radix-2^4 number are zero, it can be converted to the non-redundant radix-2^4 number by just removing the redundant bits. Also, the non-redundant numbers can be converted to the equivalent redundant numbers by attaching redundant bits $\underline{00}$ to each digit.

From the definition, the value of a d-digit redundant radix-2^r number X is up to $\sum_{i=0}^{d-1}(2^{r+2}-1)\cdot 2^{ir} = \frac{(2^{dr}-1)(2^{r+2}-1)}{2^r-1} > 2^{dr}$. However, we assume that *the valid value* of X is up to $2^{dr}-1$. If the value of X is greater than $2^{dr}-1$, it is regarded as *overflow*. For example, 4-digit redundant radix-2^4 numbers $(\underline{01}0000, \underline{00}0000, \underline{00}0000, \underline{00}0000)$ and $(\underline{00}1101, \underline{11}0000, \underline{00}0000, \underline{00}0000)$ are overflows, because their values are greater than $2^{16}-1$. We assume that, if the resulting value of an operation is a d-digit redundant radix-2^r number and it is greater than $2^{dr}-1$, it is not necessary for a circuit or a program performing the operation to guarantee the correct result due to *the overflow error*. Clearly, the redundant bits $X_{d-1}[r+1,r]$ of the most significant digit X_{d-1} of a d-digit redundant radix-2^r number X are not zero, then the value of X is overflow. Note that X can be overflow even if $X_{d-1}[r+1,r]$ is zero.

In this paper, we present hardware algorithms for various operations for redundant radix-2^r numbers. We assume that input numbers and the resulting numbers are not overflows, and the redundant bits of the most significant digit are always zero.

Table 1. The performance evaluations of our implementation and known implementation for 256-bit Montgomery modulo multiplication

	speed			hardware resources		
	freq(MHz)	cycles	time(μs)	slices	multipliers	block RAMs
Our implementation	52.86	17	0.322	2054	16	8
McIvor *et al.* [8]	76.17	93	1.22	4663	64	-
Khaleel *et al.* [1]	2.50	-	0.40	34345	-	-

3 Arithmetic Operations of Redundant/Non-redundant Numbers

3.1 Addition of Non-redundant Numbers

Let us observe the addition over non-redundant numbers. Let $X = (0101, 1010, 0101, 1001)$ and $Y = (0100, 0101, 1010, 1001)$ be 4-digit non-redundant radix-2^4 numbers. The sum $X + Y$ can be computed as follows:

$$\begin{array}{rrrrr} & 0101 & 1010 & 0101 & 1001 \\ + & 0100 & 0101 & 1010 & 1001 \\ \hline & 1010 & 0000 & 0000 & 0010 \end{array}$$

Clearly, the carry from the least significant digit is propagated to the most significant digit. We call such carry *block carry*. In other words, for two d-digit non-redundant radix-2^r numbers X and Y, if $X_0 + Y_0 \geq 2^r$, then the block carry $c_0 = 1$. Also, if $X_i + Y_i + c_{i-1} \geq 2^r$ $(1 \leq i \leq d-1)$, then the block carry $c_i = 1$. Hence, the addition has a carry chain from the least significant and the most significant digit, and it increases the delay if the addition is implemented by a combinational circuit.

3.2 Block-Carry-Free Addition for Redundant Numbers

Let us see the computation of the sum of two redundant numbers. For two 4-digit redundant radix-2^4 numbers $X = (\underline{00}0101, \underline{11}0011, \underline{11}1111, \underline{10}1111)$ and $Y = (\underline{00}0011, \underline{10}1111, \underline{01}1111, \underline{01}0001)$, their sum $Z = X + Y$ can be computed by the position sum as follows:

$$\begin{array}{rrrrr} & \underline{11} & \underline{11} & \underline{10} & \\ & 0101 & 0011 & 1111 & 1111 \\ & \underline{10} & \underline{01} & \underline{01} & \\ + & 0011 & 1111 & 1111 & 0001 \\ \hline & \underline{00}1101 & \underline{01}0110 & \underline{10}0001 & \underline{01}0000 \end{array}$$

Clearly, the addition has no block carry. Let us see the addition of two d-digit redundant radix-2^r numbers X and Y. The sum $Z = X + Y$ can be computed as follows:

$$\begin{aligned} Z_0 &= X_0[r-1,0] + Y_0[r-1,0] \\ Z_i &= X_{i-1}[r+1,r] + X_i[r-1,0] + Y_{i-1}[r+1,r] \\ &\quad + Y_i[r-1,0] \quad (1 \leq i < d) \end{aligned}$$

Hence, $Z_0 < 2^r + 2^r = 2^{r+1}$ and $Z_i < 4 + 2^r + 4 + 2^r < 2^{r+2}$ holds if $r \geq 2$. Thus, Z is a correct redundant radix-2^r number.

Let us design a combinational circuit to compute the sum $Z = X + Y$. Let $\mathrm{ADD}(2, 2, r, r)$ denote an adder circuit that computes the sum of two 2-bit and two r-bit integers. Also, let $\mathrm{ADD}(A, B, C, D)$ denote the resulting value of the sum of 2-bit numbers A and B, and r-bit numbers C and D. Clearly, $Z_0 = \mathrm{ADD}(0, 0, X_0[r-1,0], Y_0[r-1,0])$ and $Z_i = \mathrm{ADD}(X_{i-1}[r+1,r], Y_{i-1}[r+1,r], X_i[r-1,0], Y_i[r-1,0])$. Thus we have,

Lemma 1 *The addition of two d-digit redundant radix-2^r numbers can be computed using d adders $\mathrm{ADD}(2, 2, r, r)$ without block carries, whenever $r \geq 2$.*

Since the computation is performed independently in each $\mathrm{ADD}(2, 2, r, r)$, the circuit for Lemma 1 has no block carry chain. Thus, the delay time of the circuit is small and independent of d.

3.3 Block-Carry-Free Multiplication of Redundant Numbers

We show that the multiplication of 3-digit and 1-digit redundant radix-2^4 numbers can be computed without block carry. Let $X = (\underline{01}0011, \underline{10}0011, \underline{10}1111)$ and $Y = (\underline{10}0101)$. The product $X \cdot Y$ can be computed using 6-bit$\times$6-bit=12-bit multiplications as follows.

$$\begin{array}{rrrrrr} & & & \underline{01}0011 & \underline{10}1001 & \underline{01}0001 \\ \times & & & & & \underline{10}0101 \\ \hline & & & 0010 & 0111 & 0101 \\ & & 0101 & 1110 & 1101 & \\ + & 0010 & 1011 & 1111 & & \\ \hline & \underline{00}0010 & \underline{01}0000 & \underline{01}1111 & \underline{01}0100 & \underline{00}0101 \end{array}$$

Clearly, we do not have the block carries. Let us formally confirm that the multiplication of d-digit and 1-digit redundant radix-2^r numbers can be computed without block carries. Let X and Y be d-digit and 1-digit redundant radix-2^r numbers. Also, let $P_i = X_i \cdot Y$ $(0 \leq i \leq d-1)$ be the partial multiplication. Since both X_i and Y has $r+2$ bits, P_i has $2r+4$ bits. We can compute the product $S = X \cdot Y$ as follows.

$$\begin{aligned}
S_0 &= P_0[r-1,0] \\
S_1 &= P_0[2r-1,r] + P_1[r-1,0] \\
S_i &= P_{i-2}[2r+3,2r] + P_{i-1}[2r-1,r] \\
&\quad +P_i[r-1,0] \quad (2 \leq i \leq d-1) \\
S_d &= P_{d-2}[2r+3,2r] + P_{d-1}[2r-1,r] \\
S_{d+1} &= P_{d-1}[2r+3,2r]
\end{aligned}$$

Hence, $S_0 < 2^r$, $S_1 < 2^r + 2^r = 2^{r+1}$, $S_d < 2^r$, and $S_{d+1} < 2^4$ hold. Also, if $r \geq 3$ then $S_i < 2^4 + 2^r + 2^r \leq 2^{r+2}$ holds. Thus, $S = (S_{d+1}, S_d, \ldots, S_0)$ is a redundant radix-2^r number.

Let $\mathrm{MUL}(r+2, r+2)$ and $\mathrm{ADD}(4, r, r)$ denote combinational circuits to compute the $(2r+4)$-bit product of two $(r+2)$-bit numbers and the $(r+2)$-bit sum of one 4-bit and two r-bit numbers. Each of the partial products $P_{d-1} = X_{d-1} \cdot Y$, $P_{d-2} = X_{d-2} \cdot Y, \ldots, P_0 = X_0 \cdot Y$ can be computed using $\mathrm{MUL}(r+2, r+2)$. After that, each S_i can be computed using $\mathrm{ADD}(4, r, r)$. Thus, we have

Lemma 2 *The product of d-digit and 1-digit redundant radix-2^r numbers can be computed using d $\mathrm{MUL}(r+2, r+2)$s, and d $\mathrm{ADD}(4, r, r)$s, whenever $r \geq 3$.*

Next, to show a circuit to compute two d-digit redundant numbers, we will show how to add a $(d+1)$-digit radix-2^r number C to the product $X \cdot Y$. More specifically, we will show how to compute $S = X \cdot Y + C$. Later, C is used to store interim results of the product sum. We can compute each digit of the sum T can be computed as follows.

$$\begin{aligned}
T_0 &= P_0[r-1,0] + C_0[r-1,0] \\
T_1 &= P_0[2r-1,r] + P_1[r-1,0] \\
&\quad +C_0[r+1,r] + C_1[r-1,0] \\
T_i &= P_{i-2}[2r+3,2r] + P_{i-1}[2r-1,r] \\
&\quad +P_i[r-1,0] + C_{i-1}[r+1,r] \\
&\quad +C_i[r-1,0] \quad (2 \leq i \leq d-1) \\
T_d &= P_{d-2}[2r+3,2r] + P_{d-1}[2r-1,r] \\
&\quad +C_{d-1}[r+1,r] + C_d[r-1,0] \\
T_{d+1} &= P_{d-1}[2r+3,2r] + C_d[r+1,r]
\end{aligned}$$

Clearly, each T_i can be computed using $\mathrm{ADD}(2, 4, r, r, r)$, and the resulting value has no more than $r+2$ bits if $r \geq 5$. Thus, T is a $(d+2)$-digit redundant radix-2^r number and we have,

Lemma 3 *For a d-digit redundant radix-2^r number X, a 1-digit redundant radix-2^r number Y, and a $(d+1)$-digit redundant radix-2^r number C, the product sum $X \cdot Y + C$ can be computed using d $\mathrm{MUL}(r+2, r+2)$s, $d+2$ $\mathrm{ADD}(2, 4, r, r, r)$s, and a $(d+1)(r+2)$-bit registers, whenever $r \geq 5$.*

Let $T = \mathrm{PS}(X, Y, C)$ denote the circuit (or function) for Lemma 3. Using $\mathrm{PS}(X, Y, C)$ we can compute the sum C of two d-digit redundant radix radix-2^r numbers X and Y. Let $X = (X_{d-1}, X_{d-2}, \ldots, X_0)$ and $Y = (Y_{d-1}, Y_{d-2}, \ldots, Y_0)$ be two d-digit redundant radix radix-2^r numbers. We will show how to compute the product $P = (P_{2d-1}, P_{2d-2}, \ldots, P_0) = X \cdot Y$ using $\mathrm{PS}(X, Y_i, C)$. We compute partial products $X \cdot Y_0$, $X \cdot Y_1, \ldots, X \cdot Y_{d-1}$ in turn. We use $C = (C_d, C_{d-1}, \ldots, C_0)$ to denote registers storing a interim $(d+1)$-digit redundant radix radix-2^r number. We first compute $\mathrm{PS}(X, Y_0, 0)$. Then, P_0 is the least significant digit $\mathrm{PS}(X, Y_0, 0)[r+1, 0]$. We store the remaining $d+1$ digits $\mathrm{PS}(X, Y_0, 0)[(d+1)(r+2)-1, r+2]$ in C. After that, we compute $\mathrm{PS}(X, Y_1, C)$. Clearly, P_1 is the least significant digit $\mathrm{PS}(X, Y_1, C)[r+1, 0]$ holds, and then we store the remaining $d+1$ digits $\mathrm{PS}(X, Y_1, C)[(d+1)(r+2)-1, r+2]$ in C. Continuing similarly, we can obtain the product $M = X \cdot Y$. The details are spelled out as follows:

```
$C \leftarrow 0$;
for $i = 0$ to $d-1$ do
    begin
        Compute $\mathrm{PS}(X, Y_i, C)$;
        $P_i \leftarrow \mathrm{PS}(X, Y_i, C)[r+1, 0]$;
        $C \leftarrow \mathrm{PS}(X, Y_i, C)[(d+2)(r+2)-1, r+2]$;
    end
$(P_{2d-1}, P_{2d-2}, \ldots, P_d) \leftarrow (C_{d-1}, C_{d-2}, \ldots, C_0)$;
```

We have the following theorem:

Theorem 4 *For two d-digit redundant radix-2^r numbers X and Y, the product $X \cdot Y$ in the redundant radix-2^r representation can be computed in d clock cycles using d $\mathrm{MUL}(r+2, r+2)$s, $d+2$ $\mathrm{ADD}(2, 4, r, r, r)$s, and a $(d+1)(r+2)$-bit register, whenever $r \geq 5$.*

4 Montgomery Modulo Multiplication

In the RSA encryption/decryption, the modulo exponentiation $C = P^E \bmod M$ or $P = C^D \bmod M$ are computed, where P and C are plain and cypher text, and (E, M) and (D, M) are encryption and decryption keys. Usually, the number of bits in P, E, D, and M is 256 or larger. Also, the modulo exponentiation is repeatedly computed for fixed E, D, and M, and various P and C. Since modulo operation is very costly in terms of the computing time and hardware resources, we use *Montgomery modulo multiplication* [10], which does not use direct modulo operations. In Montgomery modulo multiplication, three R-bit numbers X, Y, and M are given, and $(X \cdot Y + k \cdot M) \cdot 2^{-R} \bmod M$ is computed, where an integer k is selected such that the least significant R bits of $X \cdot Y + k \cdot M$ are zero. The value of k can be computed as follows. Let $(-M^{-1})$ denote the minimum non-negative number such that $(-M^{-1}) \cdot M \equiv -1($ or $2^R - 1)$ $(\bmod\ 2^R)$. If M is odd, then $(-M^{-1}) < 2^R$ always holds. We can select k such that $k = ((X \cdot Y) \cdot (-M^{-1}))[r-1, 0]$. For such k, $(X \cdot Y + k \cdot M)[r-1, 0]$ are zero. For the reader's benefit, we will confirm this fact using an example as follows. Let $X = 10010011(147)$, $Y = 01011100(92)$, $M = 11111011(251)$, and $R = 8$. We have the product $X \cdot Y = 011010011010100(13524)$. Next, we need select an integer k such that the least significant R bits of $X \cdot Y + k \cdot M$ are zero. We have $(-M^{-1}) = 11001101(205)$, because $(-M^{-1}) \cdot M \equiv 1100100011111111(51455) \equiv -1 \pmod{2^R}$. We select $k = (X \cdot Y)[R-1, 0] \cdot (-M^{-1}) = 11000100(196)$. Then, we have the product $k \cdot M = 1100000000101100(49196)$ and the product sum $X \cdot Y + k \cdot M = 1111010100000000(62720)$. Thus, we have $(X \cdot Y + k \cdot M)[r-1, 0] = 00000000$ and $(X \cdot Y + k \cdot M) \cdot 2^{-R} = (X \cdot Y + k \cdot M)[2R-1, R] = 11110101(245)$.

Since $0 \leq X, Y < M < 2^R$ and $0 \leq k < 2^R$, we can guarantee that $(X \cdot Y + k \cdot M) \cdot 2^{-R} < 2M$. Thus, by subtracting M from $(X \cdot Y + k \cdot M) \cdot 2^{-R}$, we can obtain $(X \cdot Y + k \cdot M) \cdot 2^{-R} \bmod M$ if it is not less than M.

Since $X \cdot Y + k \cdot M \equiv X \cdot Y \pmod M$, we write $(X \cdot Y + k \cdot M) \cdot 2^{-R} \bmod M = X \cdot Y \cdot 2^{-R} \bmod M$. Let us see how Montgomery modulo multiplication is used to compute $C = P^E \bmod M$ using an example. Suppose we need to compute $C = P^E \bmod M$. For simplicity, we assume that E is a power of two. Since R and M are fixed, we can assume that $2^{2R} \bmod M$ is computed beforehand. We first compute $P \cdot (2^{2R} \bmod M) \cdot 2^R \bmod M = P \cdot 2^R \bmod M$ using the Montgomery modulo multiplication. We then compute the square $(P \cdot 2^R \bmod M) \cdot (P \cdot 2^R \bmod M) \cdot 2^{-R} \bmod M = P^2 \cdot 2^R \bmod M$. It should be clear that, by repeating the square computation using the Montgomery modulo multiplication, we have $P^E \cdot 2^R \bmod M$. After that, we multiply 1, that is $(P^E \cdot 2^R \bmod M) \cdot 1 \cdot 2^{-R} \bmod M = P^E \bmod M$ is computed. In this way, cypher text C is obtained.

4.1 Block-Carry-Free Implementation of Montgomery Modulo Multiplication

Recall that in the Montgomery modulo multiplication, R bit numbers X, Y, and M are given. In this subsection, we assume X and Y are a d-digit redundant radix-2^r number and a 1-digit redundant radix-2^r number, respectively. We will show a circuit to compute the Montgomery modulo multiplication $(X \cdot Y + k \cdot M) \cdot 2^{-r}$ for such X, Y, and M. We assume that the value of X and Y are given to the circuit as inputs, M is fixed and $(-M^{-1})$ is computed beforehand. This assumption makes sense if Montgomery modulo multiplication is used to compute the modulo exponentiation for RSA encryption and decryption.

Recall that, using the circuit for Lemma 2, $X \cdot Y$ can be computed using d $\mathrm{MUL}(r+2, r+2)$s and d $\mathrm{ADD}(4, r, r)$s. After computing $X \cdot Y$, we need to compute k such that the least significant r bits of $(X \cdot Y + k \cdot M)$ are zero. We can compute $k = ((X \cdot Y)[r-1, 0] \cdot (-M^{-1}))[r-1, 0]$ using a $\mathrm{MUL}(r, r)$. Once k is obtained, the product $k \cdot M$ is computed using the circuit for Lemma 2. Finally, the sum $(X \cdot Y + k \cdot M)$ is computed by the circuit for Lemma 1. Note that both $X \cdot Y$ and $k \cdot M$ are $(d+1)$-digit redundant radix-2^r numbers. However, since the least significant digit of $X \cdot Y$ and $k \cdot M$ are zero, we can omit the addition of the least significant digit. The readers should refer to Figure 1 for illustrating the circuit for Lemma 5.

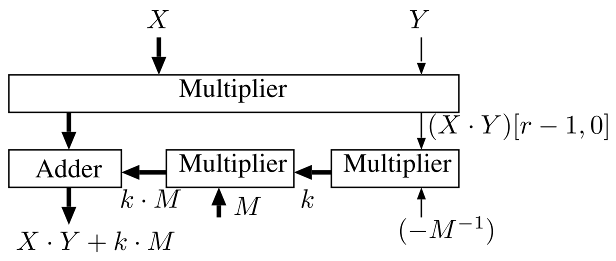


Figure 1. Circuit to compute $(X \cdot Y + k \cdot M)$ using multipliers

To compute the multiplication $X \cdot Y$, we can use a circuit for Lemma 2 which uses d $\mathrm{MUL}(r+2, r+2)$s and d $\mathrm{ADD}(4, r, r)$. To compute k, we use a $\mathrm{MUL}(r, r)$. After that to compute the multiplication $k \cdot M$, we also use a circuit for Lemma 2 and the addition $X \cdot Y + k \cdot M$ can be computed using d $\mathrm{ADD}(2, 2, r, r)$ by Lemma 1.

Therefore, we have,

Lemma 5 *Montgomery modulo multiplication $(X \cdot Y + k \cdot M) \cdot 2^{-r}$ for d-digit X and 1-digit Y of redundant radix-2^r representation can be computed using $2d+1$ $\mathrm{MUL}(r+2, r+2)$s, $2d$ $\mathrm{ADD}(4, r, r)$s, and d $\mathrm{ADD}(2, 2, r, r)$, without block carries, whenever $r \geq 4$.*

4.2 Montgomery Modulo Multiplication Using a Memory

The circuit for Lemma 5 has a cascade of three multipliers, which can be a long critical path. Also, it needs too many multipliers. We remove multipliers for computing k

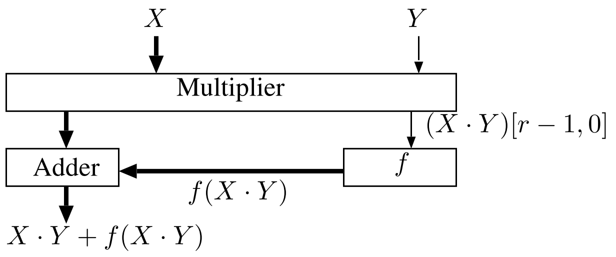


Figure 2. Circuit to compute $X \cdot Y \cdot +f(X \cdot Y)$ using a memory

to improve the circuit for Lemma 5. The key idea is to use a memory to look up the value of $k \cdot M$.

Let f be a function such that $f(Z) = (Z[r-1,0] \cdot (-M^{-1}))[r-1,0] \cdot M$. The function f can be computed using a 2^r word $(d+1)r$-bit memory as follows. The value of $f(i)$ $(0 \leq i \leq 2^r - 1)$ is stored in address i of the memory in advance. Then, by reading address $Z[r-1,0]$ of the memory, we can obtain the value of $f(Z)$ in one clock cycle. Using this memory, $f(X \cdot Y)$ can be computed in one clock cycle. After that, the addition $X \cdot Y + f(X \cdot Y)$ can be computed using $d+1$ $\mathrm{ADD}(2,2,r,r)$s from Lemma 1. Figure 2 illustrates the circuit to compute $X \cdot Y + f(X \cdot Y)$.

Note that the least significant digit of $X \cdot Y + f(X \cdot Y)$ is always zero. Hence, we can omit the computation of the least significant digit of f and the following addition. Thus, we use a 2^r word dr-bit memory for computing $f(X \cdot Y)$ and d $\mathrm{ADD}(2,2,r,r)$s to compute the sum $X \cdot Y + f(X \cdot Y)$. Therefore, we have,

Lemma 6 *Montgomery modulo multiplication $(X \cdot Y + k \cdot M) \cdot 2^{-r}$ for d-digit X and M, and 1-digit Y of redundant radix-2^r representation can be computed using d $\mathrm{MUL}(r+2, r+2)$s, $d+2$ $\mathrm{ADD}(2,4,r,r,r)$s, d $\mathrm{ADD}(2,2,r,r)$, and a 2^r-word dr-bit memory, without block carries, whenever $r \geq 5$.*

If $r = 16$ and $dr = 1024$, then the circuit for Lemma 6 needs $64k$-word 1024-bit memory of size $64M$ bits. Since the size of block memory of current FPGAs is up to few mega bits, this circuit cannot be implemented in FPGAs.

4.3 Montgomery Modulo Multiplication Using a Fewer Memory

We will reduce the size of memory to compute the function f. Recall that, k is a r-bit number such that the least significant r bits of $X \cdot Y + k \cdot M$ are zero. Let r-bit k number partition into two $r/2$ bits such that $\overline{k} = k[r-1, r/2]$ and $\underline{k} = k[r/2-1, 0]$. We can compute the values of $\overline{k}$ and $\underline{k}$ separately as follows. Let $(-M^{-1})$ be the minimum non-negative integer such that $(-M^{-1}) \cdot M \equiv -1 \pmod{2^{r/2}}$. Also, let $\overline{XY} = (X \cdot Y)[r-1, r/2]$ and $\underline{XY} = (X \cdot Y)[r/2-1, 0]$. We set $\underline{k} = \underline{XY} \cdot (-M^{-1})$. Then, the least significant $r/2$ bits of $X \cdot Y + \underline{k} \cdot M$ are zero. Let g be a function such that $g(Z) = ((Z[r/2-1,0] \cdot M)[r-1, r/2] \cdot (-M^{-1}) + c$ and $c = 0$ if $(Z \cdot M)[r/2-1, 0] = 0$ and $c = 1$ otherwise. Function g can be computed using a combinational circuit with $r/2$ input bits and $r/2$ out put bits. We set $\overline{k} = (\overline{XY} + g(\underline{XY}))[r/2-1, 0]$. Then, the least significant digit of $X \cdot Y + \underline{k} \cdot M + \overline{k} \cdot M \cdot 2^{r/2}$ is zero.

We will implement this idea in the same way as Lemma 6. Instead of computing $\underline{k}$ and $\overline{k}$, we compute $\underline{k} \cdot M$ and $\overline{k} \cdot M$ using a memory. Let h be a function such that $h(Z) = (Z[r/2-1, 0] \cdot (-M^{-1}))[r/2-1, 0] \cdot M$. Similarly to f, function g can be computed using $2^{r/2}$-word $(d(r+2) + r/2)$-bit memory. Then, $\underline{k} \cdot M = h(\underline{X \cdot Y})$ and $\overline{k} \cdot M = h(\overline{XY} + g(\underline{XY}))$ holds. Thus, $k \cdot M = \underline{k} + \overline{k} \cdot 2^{r/2} = h(\underline{X \cdot Y}) + h(\overline{XY} + g(\underline{XY})) \cdot 2^{r/2}$. The readers should refer to Figure 3 for illustrating the circuit to compute $X \cdot Y + k \cdot M = X \cdot Y + h(\underline{XY}) + h(\overline{XY} + g(\underline{XY})) \cdot 2^{r/2}$. Since a FPGAs has dual port memories, two modules to compute h in Figure 3 can be computed by a single $2^{r/2}$-word $(dr + r/2)$-bit dual port memory in the same time. The readers may think that a combinational circuit to compute g is not necessary. However, block RAMs in most FPGAs to implement a memory support only synchronous read. Thus, one clock cycle is necessary to read a memory. It follows that, if we use a memory to implement the computation of g, two clock cycles are necessary to compute $h(\overline{XY} + g(\underline{XY}))$.

The circuit to compute g is small. If $r = 16$, then a combinational circuits with 8 input bits and 8 output bits to compute g are used, and it is feasible. For example, a two-digit 7-segment decoder has 8 input bits and 14 output bits. So, necessary hardware resource to compute g is comparable to that for the two-digit 7-segment decoder. Also, we can omit the computation of the least significant $r/2$-bit of h. Thus, a single $2^{r/2}$-word dr-bit dual-port memory can compute two functions h's in the same time.

Let us evaluate the hard aware resources necessary to compute $X \cdot Y + h(\underline{X \cdot Y}) + h(\overline{X \cdot Y} + g(\underline{X \cdot Y})) \cdot 2^{r/2}$. The multiplication $X \cdot Y$ can be computed using d $\mathrm{MUL}(r+2, r+2)$s and d $\mathrm{ADD}(4,r,r)$s from Lemma 2. Function $g(\underline{X \cdot Y})$ can be computed using a combinational circuits with 8 input bits and 8 output bits and addition $\overline{X \cdot Y} + g(\underline{X \cdot Y})$ can be computed $\mathrm{ADD}(r/2, r/2)$. After that the value of function h for two arguments can be computed using a $2^{r/2}$-word dr-bit dual-port memory. Finally, the sum $(X \cdot Y) + h(\underline{X \cdot Y}) + h(\overline{X \cdot Y} + g(\underline{X \cdot Y})) \cdot 2^{r/2}$ can be computed using d $\mathrm{ADD}(2,2,2,r,r,r)$ by straightforward generalization of Lemma 1. Consequently, we have,

Lemma 7 *Montgomery modulo multiplication $(X \cdot Y + k \cdot M) \cdot 2^{-r}$ for d-digit X and M, and Y and 1-digit Y of redundant radix-2^r representation can be computed using d*

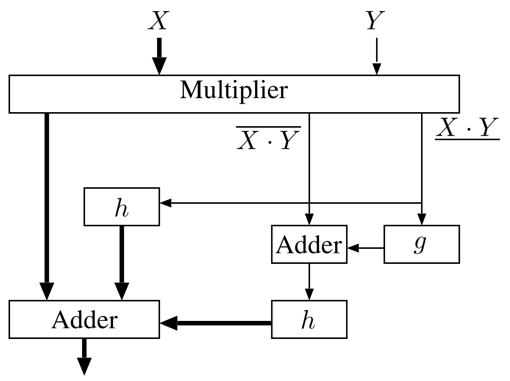
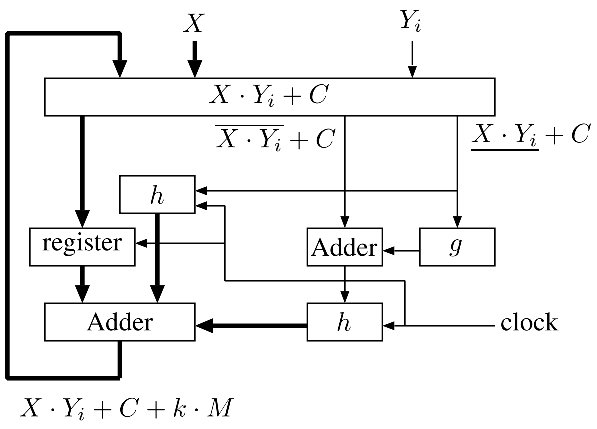


Figure 3. Circuit to compute $X \cdot Y + h(\underline{X \cdot Y}) + h(\overline{X \cdot Y} + g(\underline{X \cdot Y})) \cdot 2^{r/2}$



X
Y_i
$X \cdot Y_i + C$
$\overline{X \cdot Y_i} + C$
$\underline{X \cdot Y_i} + C$
h
register
Adder
g
Adder
h
clock
$X \cdot Y_i + C + k \cdot M$

Figure 4. Circuit to compute $X \cdot Y_i + C + k \cdot M$

$\mathrm{MUL}(r+2, r+2)$*s, d* $\mathrm{ADD}(4, r, r)$*s, one* $\mathrm{ADD}(r/2, r/2)$*, d* $\mathrm{ADD}(2, 2, 2, r, r, r)$*, a* $2^{r/2}$*-word dr-bit dual-port memory, and a combinational circuit with r/2-bit input and r/2-bit output, without block carries, whenever r* ≥ 4*.*

4.4 Montgomery Modulo Multiplication for Two d-digit Numbers

Recall that the multiplication of d-digit and 1-digit numbers are shown in Lemma 2. we have extended Lemma 2 to Theorem 4 which shows the computation of multiplication of two d-digit numbers. The same technique can be used to extend Lemma 7 to compute the Montgomery modulo multiplication of d-digit redundant radix-2^r numbers. In other words, we compute partial products $X \cdot Y_0$, $X \cdot Y_1$, ..., $X \cdot Y_{d-1}$ in turn, and compute their sum, which is equal to $X \cdot Y$.

Suppose that, as shown in Theorem 4, we use $(d+1)(r+2)$ bit register C to store $(d+1)$-bit redundant radix-2^r numbers as an interim result. To compute $X \cdot Y_i + C$ we use a circuit for Lemma 3. After that, the value of $k \cdot M$ such that the least significant digit of $X \cdot Y_i + C + k \cdot M$ is zero is obtained using the circuit for Lemma 7. Note that it uses a memory to compute function h. Thus, one clock cycle is necessary to compute $k \cdot M$. Additional one clock is necessary to store the resulting value of $X \cdot Y_i + C + k \cdot M$ in C. This implementation requires two clock cycles to compute $(X \cdot Y_i + C + k \cdot M) \cdot 2^{-r}$ and store it in C.

We can reduce these two clock cycles into one as follows. As illustrated in Figure 4, the output $X \cdot Y_i + C$ is stored in register instead of storing C. Then, the following adder computes $X \cdot Y_i + C + k \cdot M$. In this way, the value of $X \cdot Y_i + C + k \cdot M$ can be done in one clock cycle. The readers should refer to Figure 4 for illustrating the circuit.

Let us evaluate necessary hardware resources. To compute $X \cdot Y_i + C$ we use a circuit for Lemma 3, which uses d $\mathrm{MUL}(r+2, r+2)$s, $d+2$ $\mathrm{ADD}(2, 4, r, r, r)$s. As before, function g can be computed using a combinational circuit with 8 input bits and 8 output bits and function h can be computed using a dr-bit $2^{r/2}$-word dual-port memory. Also, we need one $\mathrm{ADD}(r/2, r/2)$ to compute $\overline{X \cdot Y} + g(\underline{X \cdot Y})$, and $(d+1)(r+2)$-bit register to store the value of $X \cdot Y_i + C$. After that, the sum $(X \cdot Y) + h(\underline{X \cdot Y}) + h(\overline{X \cdot Y} + g(\underline{X \cdot Y})) \cdot 2^{r/2}$ can be computed using d $\mathrm{ADD}(2, 2, 2, r, r, r)$s. Thus, we have

Theorem 8 *Montgomery modulo multiplication* $(X \cdot Y + k \cdot M) \cdot 2^{-dr}$ *for three d-digit redundant radix-*2^r *numbers X, Y and M can be computed in d clock cycles using d* $\mathrm{MUL}(r+2, r+2)$*s, d + 2* $\mathrm{ADD}(2, 4, r, r, r)$*s, d* $\mathrm{ADD}(2, 2, 2, r, r, r)$*, a* $2^{r/2}$*-word dr-bit dual port memory, and a combinational circuit with r/2-bit input and r/2-bit output, and a* $(d+1)(r+2)$*-bit register, without block carries, whenever r* ≥ 5*.*

5 Experimental Results

We have evaluated the performance of redundant radix-2^r circuits using Virtex II Pro Family FPGA XC2VP100-6, which has 99,216 slices, 444 18-bit multipliers, and 444 18k-bit dual-port block RAMs. We have used XST in ISE Foundation 9.2i for logic synthesis and analysis. Since this FPGA has 18-bit multipliers as building blocks, it makes sense to let $r = 16$. Thus, we use redundant radix-64k (i.e. radix-2^{16}) number system.

Table 2 shows the experimental results of Montgomery modulo multiplication for d-digit redundant radix-64k numbers shown in Theorem 8. In addition to the circuit for Theorem 8, the experimental results include the circuit to subtract M from the resulting value $(X \cdot Y + k \cdot M) \cdot 2^{-dr}$ to guarantee that it is smaller than M. Thus, the circuit runs in $d + 1$ clock cycles. For example, for $dr = 1024$-bit in-

Table 2. Montgomery modulo multiplication of two d-digit redundant/non-redundant radix-64k numbers

	bits	64	128	256	512	1024
	clock cycles	5	9	17	33	65
redundant	clock(MHz)	53.10	52.74	52.86	52.48	52.95
	time (μs)	0.094	0.171	0.322	0.629	1.23
	slices	560	1067	2054	3990	7883
	multipliers	4	8	16	32	64
	block RAMs	2	4	8	15	29
non-redundant	clock(MHz)	54.36	47.31	37.66	27.05	17.16
	time (μs)	0.092	0.190	0.451	1.22	3.79
	slices	453	740	1363	2574	4958
	multipliers	4	8	15	31	61
	block RAMs	2	4	8	15	29

put, it takes 65 clock cycles to complete the computation. The table also shows the experimental results of the circuits obtained by changing those for Theorem 8 to use the non-redundant number system. The experimental results show that the clock frequency of the circuits for the redundant number system is constant for every number of bits. On the other hand, the clock frequency for non-redundant number system decreases as the number of bits increases. The speed up factor of our hardware algorithm using the redundant number system over those using the conventional number system is 3 for 1024-bit Montgomery modulo multiplication.

6 Conclusions

We have introduced redundant radix-2^r number system for Montgomery modulo multiplication and its implementation on the FPGA. Our implementation for 256-bit Montgomery modulo multiplication is four times faster using less hardware resources in the FPGA than the previously know implementation.

References

[1] O. Al-Khaleel, C. Papachristou, F. Wolff, and K. Pekmestzi. FPGA-based design of a large moduli multiplier for public-key cryptographic systems. In *Proc. of International Conference on Computer Design*, pages 314 – 319, 2007.

[2] T. Blum and C. Paar. High-radix montgomery modular exponentiation on reconfigurable hardware. *IEEE Trans. on Computers*, 50(7):759–764, 2001.

[3] J. L. Bordim, Y. Ito, and K. Nakano. Accelerating the CKY parsing using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):803–810, May 2003.

[4] J. L. Bordim, Y. Ito, and K. Nakano. Instance-specific solutions to accelerate the CKY parsing for large context-free grammars. *International Journal on Foundations of Computer Science*, pages 403–416, 2004.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[6] C. K. Koc, T. Acar, and B. S. Kaliski, Jr. Analyzing and comparing Montgomery multiplication algorithms — assessing five algorithms that speed up modular exponentiation, the most popular method of encrypting and signing digital data. *IEEE Micro*, 16(3):26–33, 1996.

[7] R. Lin, K. Nakano, S. Olariu, M. C. Pinotti, J. L. Schwing, and A. Y. Zomaya. Scalable hardware-algorithms for binary prefix sums. *IEEE Trans. on Parallel and Distributed Systems*, 11(8):838–850, August 2000.

[8] C. McIvor, M. McLoone, and J. McCanny. FPGA Montgomery multiplier architectures - a comparison. In *Proc. of Field-Programmable Custom Computing Machines*, pages 279 – 282, 2004.

[9] P. V. A. Mohan. Fast algorithm for implementation of montgomery's modular multiplication technique. *Circuit System Signal Processing*, 23(6):463–478, 2004.

[10] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[11] K. Nakano and E. Takamichi. An image retrieval system using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):811–818, May 2003.

[12] K. Nakano and Y. Yamagishi. Hardware n choose k counters with applications to the partial exhaustive search. *IEICE Trans. on Information & Systems*, 2005.

[13] B. Parhami. *Computer Arithmetic - Algorithm and Hardware Designs*. Oxford University Press, 2000.

[14] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120 – 126, 1978.