Modular Multiplication of Large Integers on FPGA

Rachid Beguenane*, Jean-Luc Beuchat†, Jean-Michel Muller†, and Stéphane Simard*
*ERMETIS – Département des Sciences Appliquées
Université du Québec à Chicoutimi
555, Boulevard de l'Université
Chicoutimi (QC), G7H 2B1, Canada
†Projet Arénaire
UMR CNRS – ÉNS Lyon – UCB Lyon – INRIA 5668
46, Allée d'Italie
69364 Lyon Cedex 07, France

***Abstract*— Public key cryptography often involves modular multiplication of large operands (160 up to 2048 bits). Several researchers have proposed iterative algorithms whose internal data are carry-save numbers. This number system is unfortunately not well suited to today's Field Programmable Gate Arrays (FPGAs) embedding dedicated carry logic.**

We propose to perform modular multiplication in a high-radix carry-save number system, where the *sum bit* of the well-known carry-save representation is replaced by a *sum word*. Two digits are then added by means of a small Carry-Ripple Adder (CRA). The originality of our approach is to analyze the modulus in order to select the most efficient high-radix carry-save representation.

I. Introduction

This paper is devoted to the study of modular multiplication of large operands on Field Programmable Gate Arrays (FPGAs), which is a crucial operation in many public key cryptosystems (Elliptic Curve Cryptography, XTR, RSA). In order to compute $\langle XY \rangle_M = XY \bmod M$, where M is an n-bit integer such that $2^{n-1} < M < 2^n$, our algorithm is described by an iterative procedure based on Horner's rule:

$$\langle XY \rangle_M = \langle (\ldots((x_{r-1}Y)2 + x_{r-2}Y)2 + \ldots)2 + x_0 Y \rangle_M ,$$

where X is an unsigned r-bit number and Y is an n-bit number belonging to $\{0, \ldots, M-1\}$. This equation can be expressed recursively as follows:

$$Q[i] = \langle 2Q[i+1] + x_i Y \rangle_M , \qquad (1)$$

where $Q[r] = 0$ and $Q[0] = \langle XY \rangle_M$. Since $Q[i+1]$ and Y are less than or equal to $M-1$, $Q[i] < 3M$ and Equation (1) is implemented by means of a left shift, an addition, and up to two subtractions to perform the modulo M reduction [1].

Several improvements of the algorithm sketched by Equation (1) have been proposed. The basic idea consists in computing a number congruent with $Q[i]$ modulo M, which requires less hardware than a modulo M addition. Koç and Hung proposed a carry-save algorithm based on a sign estimation technique [2], [3]. At each step $-M$, 0, or M is added to $2Q[i+1] + x_i Y$ according to a few most significant digits of $Q[i+1]$. When M is known at design time, which is often the case in public key cryptography, another method consists in building a table $\psi(a) = \left\langle a \cdot 2^{\beta} \right\rangle_M$ and in defining the following iteration:

$$\begin{aligned} T[i] &= 2P[i+1] + x_i Y, & (2) \\ P[i] &= \psi(T[i] \text{ div } 2^{\beta}) + \langle T[i] \rangle_{2^{\beta}} , & (3) \end{aligned}$$

where $P[r] = 0$ and β is generally chosen equal to n or $n-1$. Carry-save implementations of Equations (2) and (3) have for instance been proposed by Jeong and Burleson [4], Kim and Sobelman [5], and Peeters *et al.* [6]. These algorithms are well-suited for ASIC design. However, carry-save arithmetic is expensive on Xilinx FPGAs (Virtex and Spartan families): a Full-Adder (FA) cell described in VHDL by its logic equations requires two Look-Up Tables (LUTs) to compute the sum bit and the carry bit, whereas a single LUT and the dedicated carry logic implement an FA cell of a Carry-Ripple Adder (CRA). It is of course possible to write a low level description of an FA cell based on libraries provided by Xilinx to take advantage of the carry logic. Since each slice embeds a single carry input, we can implement a single FA cell with this approach. The second LUT can only be used to implement the $\psi(a)$ table or a control unit. Though reducing the number of LUTs in the design, this solution leads to larger operators (Table I).

TABLE I
AREA AND NUMBER OF LUTS OF THREE CARRY-SAVE ITERATION STAGES.

Algorithm	Without carry logic		With carry logic	
	$n = 32$	$n = 64$	$n = 32$	$n = 64$
Jeong and Burleson [4]	107 slices 200 LUTs	210 slices 392 LUTs	119 slices 141 LUTs	232 slices 271 LUTs
Kim and Sobelman [5]	74 slices 139 LUTs	166 slices 268 LUTs	93 slices 123 LUTS	188 slices 249 LUTs
Peeters *et al.* [6]	74 slices 140 LUTs	160 slices 271 LUTs	95 slices 95 LUTs	190 slices 190 LUTs

Beuchat and Muller proposed a family of radix-2 algorithms designed for FPGAs embedding 4-input LUTs and dedicated carry logic [7] (Algorithm 1). The resulting operators are efficient for moduli up to 32 bits. Since the computation of $\psi(a)$ only requires three bits, the reduction table can be stored within the LUTs of the second CRA. However, operators based on carry-save adders are much faster for larger operands (Table II).

Algorithm 1 Radix-2 modulo M multiplication [7].

```
Require: An r-bit number $X \in \mathbb{N}$, $0 < Y \leq Y_{\max} < (2^{n+2} + 11 - 4 \cdot (n \bmod 2))/3$, and $\psi(a) = \left\langle a \cdot 2^{n-1} \right\rangle_M$, $\forall a \in \{0, \ldots, 7\}$
Ensure: $P[0] \leq Y_{\max}$ and $P = \langle XY \rangle_M$
 1: $P[r] \leftarrow 0$;
 2: for $i$ in $r-1$ downto $0$ do
 3:     $T[i] \leftarrow 2P[i+1] + x_i Y$;
 4:     $P[i] \leftarrow \psi(T[i] \text{ div } 2^{n-1}) + \langle T[i] \rangle_{2^{n-1}}$;
 5: end for
 6: if $P[0] \geq M$ then
 7:     $P \leftarrow P[0] - M$;
 8: else
 9:     $P \leftarrow P[0]$;
10: end if
```

In this paper, we describe an implementation of Algorithm 1 in a high-radix carry-save number system, which is briefly reviewed in Section II. The basic idea consists in replacing the *sum bits* of the carry-save representation by *sum words* which are efficiently added by means of small CRAs. Our high-radix carry-save modular multiplication algorithm is described in Section III. Its main originality is to analyze the modulus M in order to choose the best high-radix carry-save representation of $P[i]$ and $T[i]$. Consequently, both architecture and performance of our operators depend on M. We wrote a VHDL code generator to easily build several modulo M multipliers and to compare them against previously published solutions (Section IV). Finally, we conclude in Section V.

TABLE II
AREA AND DELAY OF CARRY-SAVE AND RADIX-2 ITERATION STAGES.

Algorithm	$n = 16$	$n = 32$	$n = 64$
Jeong and Burleson [4]	58 slices 9 ns	127 slices 11 ns	236 slices 14 ns
Kim and Sobelman [5]	41 slices 8 ns	79 slices 10 ns	150 slices 12 ns
Peeters *et al.* [6]	50 slices 8 ns	86 slices 11 ns	163 slices 12 ns
Beuchat and Muller [7]	21 slices 12 ns	40 slices 14 ns	80 slices 20 ns

II. High-Radix Carry-Save Numbers

Definition 1: A k-digit high-radix carry-save number X is denoted by

$$X = (x_{k-1}, \ldots, x_0) = \left(\left(x_{k-1}^{(c)}, x_{k-1}^{(s)} \right), \ldots, \left(x_0^{(c)}, x_0^{(s)} \right) \right),$$

where the jth digit x_j consists of an n_j-bit sum word $x_j^{(s)}$ and a carry bit $x_j^{(c)}$ such that $x_j = x_j^{(s)} + x_j^{(c)} 2^{n_j}$. Note that this representation does not require a fixed radix. We have:

$$\begin{aligned} X &= x_0 + x_1 2^{n_0} + x_2 2^{n_0+n_1} + \ldots + x_{k-1} 2^{n_0+\ldots+n_{k-2}} \\ &= x_0^{(s)} + \sum_{i=0}^{k-2} \left(x_i^{(c)} + x_{i+1}^{(s)} \right) 2^{\sum_{j=0}^{i} n_i} + x_{k-1}^{(c)} 2^{\sum_{j=0}^{k-1} n_i}. \end{aligned}$$

Example 1: Consider the 5-digit high-radix carry-save number $X = ((5,1),(0,0),(3,1),(6,1),(4,0))$ with $n_4 = n_3 = n_0 = 3$, $n_2 = 2$, and $n_1 = 4$ (Figure 1). According to Definition 1, X represents 54324:

$$\begin{aligned} X \;=\; & 4 + (0+6) \cdot 2^3 + (1+3) \cdot 2^7 + (1+0) \cdot 2^9 + \\ & (0+5) \cdot 2^{12} + 1 \cdot 2^{15} = 54324. \end{aligned}$$

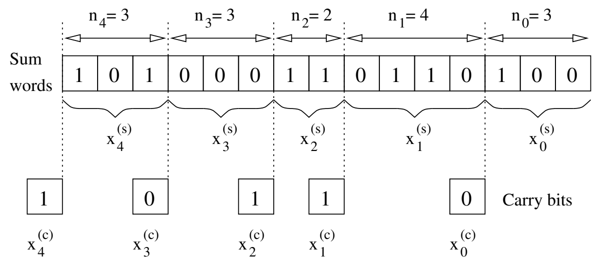

Fig. 1. High-radix carry-save number.

III. High-Radix Carry-Save Modular Multiplication

A. High-Radix Carry-Save Iteration Stage

Assume that $P[i+1]$ is a k-digit high-radix carry-save number whose sum words are n_0-, $\ldots$, n_{k-1}-bit unsigned integers such that $n_0 + \ldots + n_{k-1} = n$. The addition of $2P[i+1]$ and the unsigned binary number $x_i Y$ can be performed by means of k CRAs, thus taking advantage of the dedicated carry logic available in today's FPGAs. The first digit of $2P[i+1]$ is constituted of the (n_0+1)-bit word $2p_0^{(s)}[i+1]$ and of the carry bit $p_0^{(c)}[i+1]$ of weight 2^{n_0+1}. We split $x_i Y$ into k words $w_0[i]$, $\ldots$, $w_{k-1}[i]$ whose respective widths are (n_0+1), $\ldots$, n_{k-1}. Assume that the *msb* function returns the most significant bit of its argument. Since $x_i Y$ is an n-bit number, *msb*$(w_{k-1}[i]) = 0$. Consequently, the jth digit of $T[i]$ is obtained by adding the jth sum word of $2P[i+1]$ and the jth word of $x_i Y$ with a CRA whose input carry is set to $p_{j-1}^{(c)}[i+1]$, where $0 \leq j \leq k-1$ and $p_{-1}^{(c)}[i+1] = 0$. Note that the most significant digit of $T[i]$ contains two carry bits: the output carry of the kth adder and $p_{k-1}^{(c)}[i+1]$. However, these bits will address the table responsible for the modular reduction and this special intermediate format does not require a conversion.

Example 2: Figure 2 describes an iteration stage of a 15-bit modulo M multiplier with $P[i+1] = 108648$, $x_i Y = 26006$, and $M = 32761$. The format of the high-radix number $P[i+1]$ is defined by the set $\{n_0 = 3, n_1 = 4, n_2 = 2, n_3 = 3, n_4 = 3\}$. The least significant digit of $T[i]$ is therefore constituted of an $(n_0+1) = 4$-bit sum word and a carry bit such that

$2^4 t_0^{(c)}[i] + t_0^{(s)}[i] = 8 + 6 = 14$. Since the outputs of the five CRAs are:

$$\begin{cases} t_0^{(s)}[i] = 14, & t_0^{(c)}[i] = 0, \\ t_1^{(s)}[i] = 15, & t_1^{(c)}[i] = 0, \\ t_2^{(s)}[i] = 1, & t_2^{(c)}[i] = 1, \\ t_3^{(s)}[i] = 2, & t_3^{(c)}[i] = 0, \\ t_4^{(s)}[i] = 0, & t_4^{(c)}[i] = 1, \end{cases}$$

we check that

$$\begin{aligned} &2P[i+1] + x_i Y \\ &\quad = 2 \cdot 54324 + 26006 \\ &\quad = 14 + (0+15)\cdot 2^4 + (0+1)\cdot 2^8 + (1+2)\cdot 2^{10} + \\ &\qquad (0+0)\cdot 2^{13} + (1+1)\cdot 2^{16} \\ &\quad = 134654. \end{aligned}$$

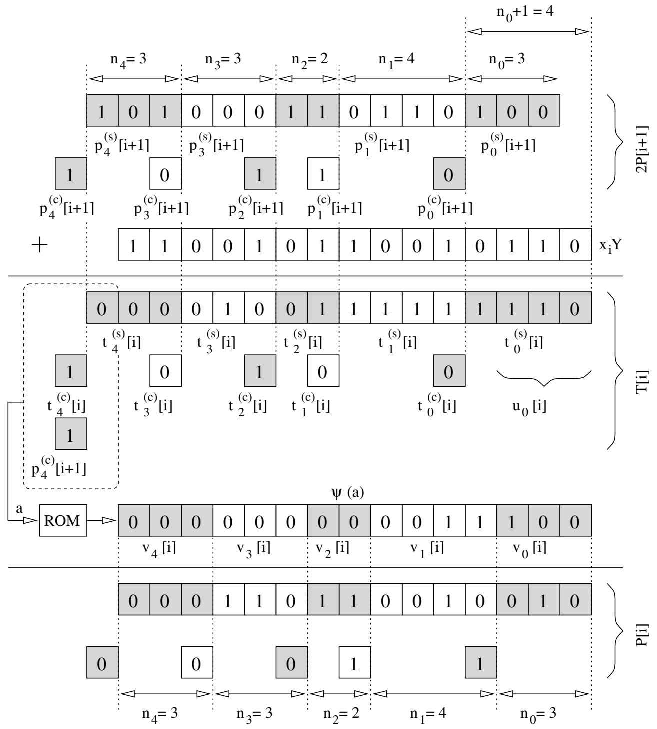

Fig. 2. Iteration stage of a high-radix carry-save modulo 32761 multiplier.

The modulo M reduction of $T[i]$ is the crucial operation of our algorithm. The most significant bit of $t_{k-1}^{(s)}[i]$ as well as the two carry bits $t_{k-1}^{(c)}[i]$ and $p_{k-1}^{(c)}[i+1]$ address a table containing multiples of 2^n modulo M. Since $msb(w_{k-1}[i]) = 0$, we have

$$\begin{aligned} &2^{n_{k-1}} t_{k-1}^{(c)}[i] + t_{k-1}^{(s)}[i] \\ &\quad = \underbrace{p_{k-1}^{(s)}[i+1]}_{\leq 2^{n_k}-1} + \underbrace{w_{k-1}[i]}_{\leq 2^{n_{k-1}}-1} + p_{k-2}^{(c)} \\ &\quad \leq 2^{n_k} + 2^{n_{k-1}} - 1. \end{aligned}$$

Thus, if $t_{k-1}^{(c)}[i] = 1$, then $msb(t_{k-1}^{(s)}[i]) = 0$ and

$$2t_{k-1}^{(c)}[i] + 2p_{k-1}^{(c)}[i+1] + msb(t_{k-1}^{(s)}[i]) \leq 4.$$

We deduce from this inequality that the table only contains the five unsigned numbers $\langle a \cdot 2^n \rangle_M$ with $a \in \{0, \ldots, 4\}$. Remember now that the least significant sum word of $T[i]$ is an $(n_0 + 1)$-bit number. The last step of our algorithm thus consists in adding the output of the table while converting the result to the format of $P[i+1]$. The main difficulty arises from the carry bits of $T[i]$ which can not be considered as input carries of the CRAs. The weight of $t_0^{(c)}[i]$ is for instance 2^{n_0+1}, whereas the one of $p_0^{(c)}[i]$ is 2^{n_0}.

Example 3 (example 2 continued): Since $t_4^{(c)}[i] = p_4^{(c)}[i+1] = 1$ and $msb(t_4^{(s)}[i]) = 0$ (Figure 2), the table responsible for the modular reduction returns $\psi(4) = \langle 2^{n+2} \rangle_{32761} = 28$. Then, we split the word made up of the n least significant sum bits of $T[i]$ into four words of n_0, n_1, n_2, and n_3 bits. We obtain $u_0[i] = 6$, $u_1[i] = 15$, $u_2[i] = 3$, $u_3[i] = 4$, and $u_4[i] = 0$. We also split $\psi(4)$ into four words $v_0[i] = 4$, $v_1[i] = 3$, and $v_2[i] = v_3[i] = v_4[i] = 0$. The first digit of $P[i]$ can be computed with a CRA. From

$$2^3 p_0^{(c)}[i] + p_0^{(s)} = u_0[i] + v_0[i] = 10,$$

we deduce that $p_0^{(c)}[i] = 1$ and $p_0^{(s)} = 2$. However, the computation of the remaining digits is defined by:

$$2^{n_j} p_j^{(c)}[i] + p_j^{(s)} = u_j[i] + v_j[i] + 2t_{j-1}^{(c)}[i],$$

where $1 \leq j < k$, and requires three-operand adders. We obtain $P[i] = ((0,0),(0,6),(0,3),(1,2),(1,2)) = 3610$ and verify that

$$134654 \equiv 3610 \pmod{32761}.$$

In the following, we propose a method to efficiently merge the carry bits and $\psi(a)$ so that the computation of $P[i]$ only requires k CRAs. The format of the high-radix carry-save numbers as well as the architecture of the operator will therefore depend on the chosen modulus M.

B. Modulo M Reduction Table

Let us consider the 32-bit modulus $M = 4294967111$ whose correction table is stored in the matrix Ψ_M:

$$\Psi_M = \begin{bmatrix} 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ldots & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & \ldots & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & \ldots & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & \ldots & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

In this example, $\psi(a)$ is a 10-bit word, $\forall a \in \{0, \ldots, 4\}$. If $n_0 = 9$, we can therefore merge the output of the table and the carry bits $t_j^{(c)}[i]$ as follows:

$$v_j[i] = \begin{cases} \sum_{q=0}^{9} \psi_q(a) & \text{if } j = 0, \\ 2t_0^{(c)}[i] + \psi_9(a) & \text{if } j = 1, \\ 2t_{j-1}^{(c)}[i] & \text{otherwise.} \end{cases}$$

Figure 3 describes the hardware architecture of an iteration stage. We assume here that $n_1 = 5$ and the critical path includes therefore a 10-bit CRA and a 5-bit CRA on Virtex or Spartan FPGAs (a dedicated AND gate is associated with each LUT to compute $x_i Y$ and each bit of $\psi(a)$ is computed within a LUT of the second CRA). It is however possible to further reduce n_0 at the price of a slightly more complex logic. Suppose for instance that $n_0 = 1$. We have to combine a carry bit $t_0^{(c)}[i]$ of weight 2^{n_0+1} with the output of the table. Adding $t_0^{(c)}[i]$ to $\psi_2(a)$ only generates a carry bit for $a = 4$. However, $\psi_3(4) = 0$ and the carry propagation is limited to one position. Consequently, if Ψ_M does not contain enough columns such that $\psi_i(a) = 0$, $\forall a \in \{0, \ldots, 4\}$, we have to identify a set of columns minimizing the carry propagation induced by the addition of $t_i^{(c)}[i]$.

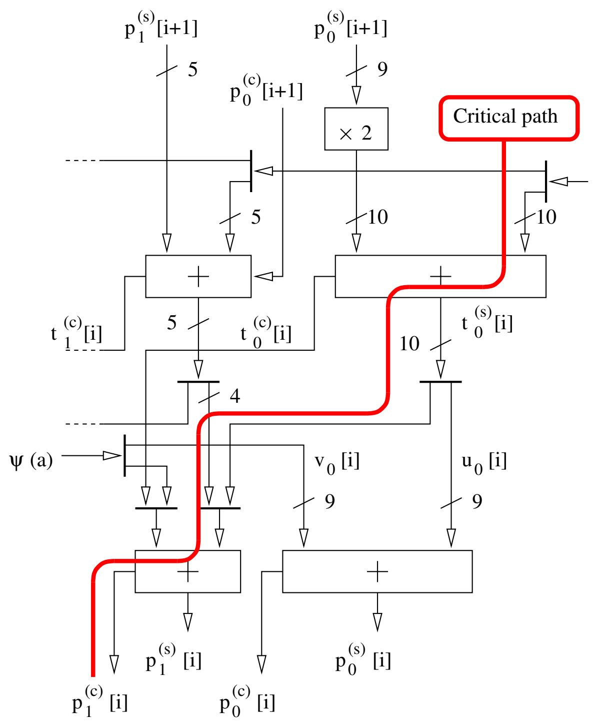

Fig. 3. Architecture of a modulo 4294967111 multiplier with $n_0 = 9$ and $n_1 = 5$.

However, these two strategies fail for several moduli whose reduction table is as follows:

$$\Psi_M = \begin{bmatrix} \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & \cdots \\ \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \\ \cdots & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & \cdots \end{bmatrix}.$$

Let us consider the nine non-zero columns of this matrix. We define

$$\psi'(a) = \begin{cases} 0 & \text{if } a = 0, \\ 2^9 - 1 & \text{if } 1 \le a \le 4. \end{cases}$$

A HA cell computes $\psi'_0(a) + t_j^{(c)}[i]$ and returns a sum and a carry bit respectively denoted by $\sigma^{(s)}$ and $\sigma^{(c)}$. Consequently, we have

$$\psi'(a) + t_j^{(c)}[i] = \begin{cases} \sigma^{(s)} & \text{if } a = 0, \\ 2^9 - 1 + \sigma^{(s)} = \sigma^{(c)} 2^9 + \sum_{i=1}^{8} \bar{\sigma}^{(c)} + \sigma^{(s)} & \text{if } 1 \le a \le 4. \end{cases}$$

Therefore, the modification of the reduction table only requires a HA cell, an inverter, and an AND gate:

$$\psi'(a) + t_j^{(c)}[i] = \sigma^{(c)} 2^9 + \sum_{i=1}^{8} \psi'_0(a) \bar{\sigma}^{(c)} + \sigma^{(s)}.$$

We can apply the same strategy to combine several carry bits $t_{j+q}^{(c)}$ with $\psi'(a)$ (Figure 4). Algorithm 2 summarizes the high-radix carry-save modular multiplication method proposed in this paper.

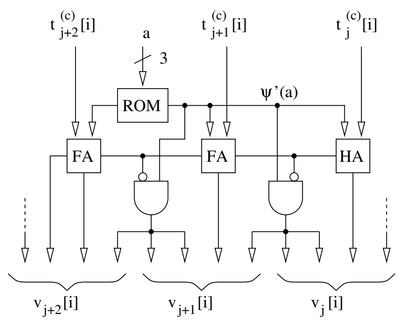

Fig. 4. Modification of the reduction table Ψ_M when $\psi_j(a) = 1$, $1 \le a \le 4$, for several successive values of j.

IV. Implementation Results

In order to compare our algorithm against the one proposed by Peeters *et al.* [6], which is to our knowledge the best previously published modular multiplier based on Horner's rule, we generated 250 prime numbers of 64, 128, 192, and 256 bits. In all experiments, the maximum width of a sum word was 8 bits. We selected moduli for which the table Ψ_M contains almost only 1s. Such numbers are considered as worst-case moduli because they require two levels of logic (a CRA and AND gates) to combine the carry bits and the table (see Figure 4). For such moduli, our approach allows to reduce the area by 35 to 40% (Figure 5) at the price of a slightly larger critical path (Figure 6)[1]. For moduli whose table Ψ_M is a sparse matrix, our first experiments indicate that our method roughly divide by two the number of slices of an iteration stage without increasing the critical path. However, we have to consider a larger set of moduli to confirm these results.

[1]We found only a few moduli for which our method significantly increases the critical path. There is for instance a single 256-bit modulus whose delay is 15.3 ns. It should be possible to shorten the critical path by reducing the maximal width of sum words.

Algorithm 2 High-radix carry-save modulo M multiplication.

```
Require: An r-bit number $X \in \mathbb{N}$, $Y \in \{0,\dots,M-1\}$,
    $\psi(a) = \langle a \cdot 2^n \rangle_M$, $\forall a \in \{0,\dots,4\}$, the function $msb(Z)$
    which returns the most significant bit of $Z$, the function
    $merge(\psi(a), t^{(c)}_{k-2}[i], \dots, t^{(c)}_0[i])$ which combines the output
    of the correction table and $k-1$ carry bits, and the set
    $\{n_0,\dots,n_{k-1}\}$ which defines the format of the high-radix
    carry-save numbers $P[i]$ and $T[i]$
Ensure: $P = \langle XY \rangle_M$
 1: $P[r] \leftarrow 0$;
 2: for $i$ in $r-1$ downto $0$ do
 3:     Split $x_iY$ into $k$ words $w_0[i]$, $w_1[i]$, $\dots$, $w_{k-2}[i]$,
        $w_{k-1}[i]$ of respective widths $(n_0+1)$, $n_1$, $\dots$, $n_{k-2}$,
        $(n_{k-1}-1)$;
 4:     $2^{n_0+1}t^{(c)}_0[i] + t^{(s)}_0[i] \leftarrow 2p^{(s)}_0[i+1] + w_0[i]$;
 5:     for $j$ in $1$ to $k-1$ do
 6:         $2^{n_j}t^{(c)}_j[i] + t^{(s)}_j[i] \leftarrow p^{(s)}_j[i+1]) + w_j[i] + p^{(c)}_{j-1}[i+1]$;
 7:     end for
 8:     $U[i] \leftarrow n$ least significant sum bits of $T[i]$;
 9:     $a \leftarrow 2p^{(c)}_{k-1}[i+1] + 2t^{(c)}_{k-1}[i] + msb(t^{(s)}_{k-1}[i])$;
10:     $V[i] \leftarrow merge(\psi(a), t^{(c)}_{k-2}[i], \dots, t^{(c)}_0[i])$;
11:     Split $U[i]$ and $V[i]$ into $k$ words of respective widths
        $n_0, \dots, n_{k-1}$;
12:     for $j$ in $0$ to $k-1$ do
13:         $2^{n_j}p^{(c)}_j[i] + p^{(s)}_j[i] \leftarrow u_j[i] + v_j[i]$;
14:     end for
15: end for
16: $P \leftarrow \langle p_0[0] + p_1[0]2^{n_0} + \dots + p_{k-1}[0]2^{n_0+\dots+n_{k-2}} \rangle_M$;
```

V. Conclusion

In this paper, we have proposed a high-radix carry-save FPGA implementation of modular multiplication. Our results indicate that this approach significantly reduces the area of the iteration stage compared to previously published solutions, while only slightly increasing the critical path for some moduli.

Our algorithm returns a high-radix carry-save number $P[0]$ which we have to convert to standard binary representation. This operation requires a modular adder whose architecture depends on the modulus M. The second addition of the iteration stage involves the n least significant sum bits of $T[0]$ and $(k-1)$ carry-bits combined with $\psi(a)$. Therefore,

$$P[0] \le 2^n - 1 + 2^{n_0+\dots+n_{k-2}+1} + \dots + 2^{n_0+1} + \max_{1\le a\le 4}(\psi(a)).$$

We can use this equation to find an integer q such that $P[0] < qM$. The conversion is however expensive if $q \ge 2$. We plan to design an algorithm which guarantees that $P[0] = \langle XY \rangle_M$ or $P[0] = \langle XY \rangle_M + M$.

References

[1] G. R. Blakley, "A computer algorithm for calculating the product ab modulo m," *IEEE Trans. Comput.*, vol. C–32, no. 5, pp. 497–500, 1983.

[2] C. K. Koç and C. Y. Hung, "Carry-save adders for computing the product AB modulo N," *Electronics Letters*, vol. 26, no. 13, pp. 899–900, June 1990.

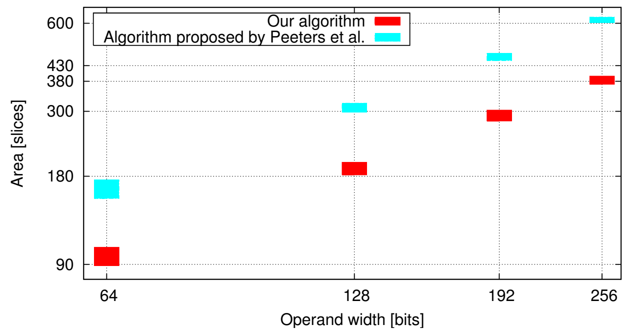

Fig. 5. Area comparison for worst-case moduli (place-and-route results, Spartan-3 FPGA).

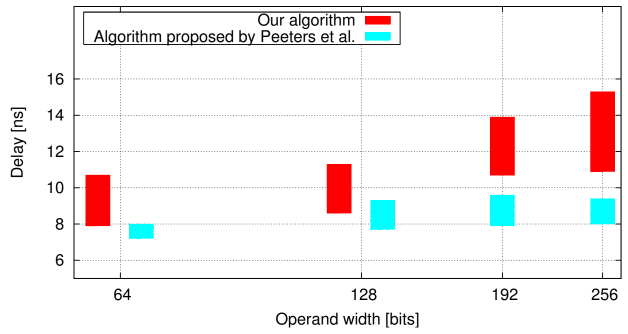

Fig. 6. Delay comparison for worst-case moduli (place-and-route results, Spartan-3 FPGA).

[3] ——, "A fast algorithm for modular reduction," *IEE Proceedings: Computers and Digital Techniques*, vol. 145, no. 4, pp. 265–271, July 1998.

[4] Y.-J. Jeong and W. P. Burleson, "VLSI array algorithms and architectures for RSA modular multiplication," *IEEE Trans. VLSI Syst.*, vol. 5, no. 2, pp. 211–217, June 1997.

[5] S. Kim and G. E. Sobelman, "Digit-serial modular multiplication using skew-tolerant domino CMOS," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2. IEEE Computer Society, 2001, pp. 1173–1176.

[6] E. Peeters, M. Neve, and M. Ciet, "XTR implementation on reconfigurable hardware," in *Cryptographic Hardware and Embedded Systems – CHES 2004*, ser. Lecture Notes in Computer Science, M. Joye and J.-J. Quisquater, Eds., no. 3156. Springer, 2004, pp. 386–399.

[7] J.-L. Beuchat and J.-M. Muller, "Modulo m multiplication-addition: Algorithms and FPGA implementation," *Electronics Letters*, vol. 40, no. 11, pp. 654–655, May 2004.

Acknowledgment

J.-L. Beuchat is funded by Swiss National Science Foundation fellowship for advanced researchers PA002–101386.