Power Attack Resistant Efficient FPGA Architecture for Karatsuba Multiplier

Chester Rebeiro[1] and Debdeep Mukhopadhyay[2]
[1] MS Scholar, Dept. of Computer Science and Engineering
[2] Assistant Professor, Dept. of Computer Science and Engineering
Indian Institute of Technology Madras, India
{rebeiro, debdeep}@cse.iitm.ernet.in

Abstract

The paper presents an architecture to implement Karatsuba Multiplier on an FPGA platform. Detailed analysis has been carried out on how existing algorithms utilize FPGA resources. Based on the observations the work develops a hybrid technique which has a better area delay product compared to the known algorithms. The results have been practically demonstrated through a large number of experiments. Subsequently, the work develops a masking strategy to prevent power based side channel attacks on the multiplier. It has been found that the proposed masked Hybrid Karatsuba multiplier is more compact compared to existing designs.

1 Introduction

Elliptic Curve Cryptography (ECC) is becoming increasing popular for public key cryptosystems because it offers more security per key bit than any other crypto algorithm. This results in shorter keys and smaller implementations. ECC implementation follows a layered hierarchical scheme. The lower layers in the hierarchy are the finite field arithmetic operations, while the top layers are the scalar multiplications in the group formed by the Elliptic Curves. The performance of the top layers of the hierarchy is greatly influenced by the performance of the underlying layers. It is therefore important to have efficient implementations of finite field arithmetic. In this paper we discuss an important component of the ECC implementation: the finite field multiplier.

There are several methods to implement finite field multiplication, however the Karatsuba multiplication [4] algorithm is the most efficient. The basic Karatsuba algorithm is ideally suited when the number of bits (n) of the multiplicands is a power of 2. i.e. $n = 2^k$. If $n \neq 2^k$, as in the finite fields used for ECC, an adaption of the Karatsuba algorithm has to be used. There have been several research works [6][11][3][9] for Karatsuba multipliers in such fields.

In this paper we present a novel Hybrid Karatsuba multiplier for elliptic curves. Our multiplier is designed specifically for FPGA platforms and requires least resources as compared to other Karatsuba multipliers used for elliptic curve arithmetic. We selected FPGAs as our development platform because of the low non recurring costs, simpler design cycles, faster time to market, greater performance per unit area and the rapid growth in FPGA usage. Besides these, FPGAs for cryptography applications have several advantages over ASICs[12]. However there has been little work done on the security aspects of FPGAs. It has been shown in [8] that Side Channel Attacks (SCAs) on FPGAs is a reality, therefore cryptography implementations on the FPGA have to be made resistant to such attacks. In this paper we develop an efficient technique to prevent Differential Power Analysis (DPA) [5] on our multiplier.

In section 2 we present the background for finite field multiplication and side channel attacks. Section 3 discusses how the existing Karatsuba algorithms get mapped on to an FPGA. Section 4 presents our proposed Karatsuba multiplier. Section 5 shows how our multiplier can be made resistant to DPA attacks. Section 6 has the conclusion.

2 Preliminaries

In this section we present a background on the topic of finite field multiplication and the Karatsuba method to multiply efficiently. We also briefly state about Side Channel Attacks (SCA) on cryptographic algorithms.

2.1 Finite Field Multiplications and the Karatsuba Algorithm

Finite field multiplication of two elements in the field $GF(2^n)$ is defined as

$$C(x) = A(x)B(x) mod P(x) \quad (1)$$

where $A(x)$, $B(x)$ and $C(x) \in GF(2^n)$, and $P(x)$ is the irreducible polynomial of degree n which generates the field

1063-9667/08 $25.00 © 2008 IEEE
DOI 10.1109/VLSI.2008.65

$GF(2^n)$. Implementing the multiplication requires two steps. First, the polynomial product $C'(x) = A(x)B(x)$ is determined, then, the modulo operation is done on $C'(x)$. The Karatsuba Algorithm is used for the polynomial multiplication. The Karatsuba multiplier achieves its efficiency by splitting the n bit multiplicands into two 2-term polynomials as shown in equation (2).

$$A(x) = A_h x^{n/2} + A_l \qquad B(x) = B_h x^{n/2} + B_l \tag{2}$$

The multiplication is then done using three $n/2$ bit multiplications as shown in equation (3). The three $n/2$ bit multiplications are implemented recursively using the Karatsuba algorithm.

$$\begin{aligned} C'(x) &= (A_h x^{n/2} + A_l)(B_h x^{n/2} + B_l) \\ &= A_h B_h x^n + (A_h B_l + A_l B_h) x^{n/2} + A_l B_l \\ &= A_h B_h x^n \\ &\quad + ((A_h + A_l)(B_h + B_l) + A_h B_h + A_l B_l) x^{n/2} \\ &\quad + A_l B_l \end{aligned} \tag{3}$$

The gate requirements for an n bit recursive Karatsuba multiplier is given below.

$$\begin{aligned} \#AND \text{ gates}: &\quad n^{log_2 3} \\ \#XOR \text{ gates}: &\quad \textstyle\sum_{r=0}^{log_2 n} 3^r \left(4n/2^r - 4\right) \end{aligned}$$

2.2 Karatsuba Multiplication for Elliptic Curves

The basic recursive Karatsuba multiplier cannot be applied directly to ECC, because the binary extension fields used in standards such as in [10] have a degree which is prime. There have been several published works which implement a modified Karatsuba algorithm for use in elliptic curves. The easiest method to modify the Karatsuba algorithm for elliptic curves is by padding. The *Padded Karatsuba Multiplier* extends the n bit multiplicands to $2^{\lceil log_2 n \rceil}$ bits by padding the most significant bits with zeroes. This allows the use of the basic recursive Karatsuba algorithm. The obvious drawback of this method is the extra arithmetic introduced due to the padding.

In [6], a *Binary Karatsuba Multiplier* was proposed to handle multiplications in any field of the form $GF(2^n)$, where $n = 2^k + d$, and k is the largest integer such that $2^k < n$. The Binary Karatsuba multiplier splits the n bit multiplicands (A and B) into two terms. The lower terms (A_l and B_l) have 2^k bits while the higher terms (A_h and B_h) have d bits. Two 2^k bit multipliers are required to obtain the partial products $A_l B_l$ and $(A_h + A_l)(B_h + B_l)$. For the latter multiplication, the A_h and B_h terms have to be padded by $2^k - d$ bits. $A_h B_h$ product is determined using a d bit Binary Karatsuba multiplier.

In [9], a *Recursively Applied Iterative Karatsuba* implementation was presented. In this architecture, each of the $n/2$ bit multiplications is done serially using the same hardware. This has the advantage of a reduction in the area required at the cost of increased timing requirements. However this technique still relies on padding when n is not a power of two. Although there is a marginal reduction in the number of XOR gates, the number of AND gates remains the same.

The *Simple Karatsuba Multiplier* [11] is the basic recursive Karatsuba multiplier with a small modification. If an n bit multiplication is needed to be done, n being any integer, it is split into two polynomials as in equation (3). The A_l and B_l terms have $\lceil n/2 \rceil$ bits and the A_h and B_h terms have $\lfloor n/2 \rfloor$ bits. The Karatsuba multiplication can then be done with two $\lceil n/2 \rceil$ bit multiplications and one $\lfloor n/2 \rfloor$ bit multiplication. The upper bound for the number of AND gates and XOR gates required for the Simple Karatsuba multiplier is the same as that of a $2^{\lceil log_2 n \rceil}$ bit basic recursive Karatsuba multiplier. The maximum number of gates required and the time delay for an n bit Simple Karatsuba multiplier is given below.

$$\begin{aligned} \#AND \text{ gates} &: 3^{\lceil log_2 n \rceil} \\ \#XOR \text{ gates} &: \textstyle\sum_{r=0}^{\lceil log_2 n \rceil} 3^r \left(4\lceil n/2^r \rceil - 4\right) \end{aligned}$$

In the *General Karatsuba Multiplier* [11], the multiplicands are split into more than two terms. For example an n term multiplier is split into n different terms. The number of gates required is given below.

$$\begin{aligned} \#AND \text{ gates}: &\quad n(n+1)/2 \\ \#XOR \text{ gates}: &\quad \tfrac{5}{2}n^2 - \tfrac{7}{2}n + 1 \end{aligned}$$

2.3 Side Channel Resistant Multiplication

An important aspect of the finite field multiplier used for elliptic curve cryptosystems is that they have to be resistant to SCAs. SCAs are the biggest threat to modern cryptosystems. In these attacks knowledge is gathered about the key by exploiting the information that leaks from various sources in the device. Among all side channel attacks, SCA based on power consumption of the device is the most practical and threatening. There are two techniques for power analysis: Simple Power Analysis (SPA) and Differential Power Analysis (DPA). DPA exploits the fact that power consumption of a chip depends on intermediate results of the algorithm.

One of the most common techniques to counter DPA in multipliers is by using masking [2]. The main idea behind a masked multiplier is to make all intermediate values of the multiplier independent of the multiplicands. Such multipliers are secure against power attacks, if the underlying CMOS gates switch once per clock cycle [7].

3 Karatsuba Implementation on FPGA

In this section we discuss the mapping of the recursive Karatsuba algorithm and the General Karatsuba algorithm on an FPGA. We estimate the amount of FPGA resources that is required for the Karatsuba implementations. This demands a special section because of the unique architecture of the FPGA compared with other programmable devices. The Xilinx FPGA [14] is made up of Configurable Logic Blocks (CLBs). Each CLB on a Xilinx Virtex 4 FPGA

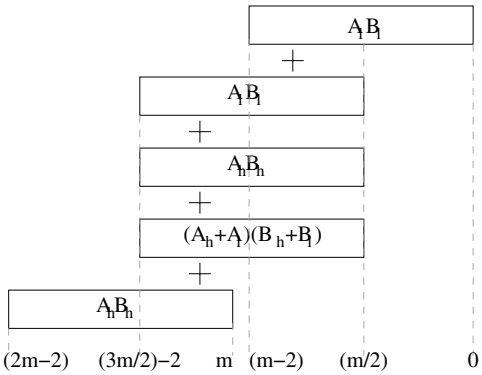

Figure 1. Combining the Partial Products

contains two slices. Each slice contains two lookup tables (LUTs). Each LUT has four inputs and can be configured for any logic function having a maximum of four inputs. The LUT can also be used to implement logic functions having less than four inputs, two for example. In this case, only half the LUT is utilized the remaining part is not utilized. Such a LUT which has less than four inputs is an under utilized LUT. For example, the logic function $y = x_1 + x_2$ under utilizes the LUT as it has only two inputs. Most compact implementations are obtained when the utilization of each LUT is maximized. From the above fact it may be derived that the minimum number of LUTs required for a q bit input is given by the equation (4):

$$\#LUT(q) = \begin{cases} 0 & \text{if } q = 1 \\ 1 & \text{if } 1 < q \leq 4 \\ \lceil q/3 \rceil & \text{if } q > 4 \text{ and } q\text{mod}3 = 2 \\ \lfloor q/3 \rfloor & \text{if } q > 4 \text{ and } q\text{mod}3 \neq 2 \end{cases} \quad (4)$$

The mapping of the algorithm on the FPGA depends on several factors in addition to the FPGA architecture such as the synthesis tool used, the synthesis options used etc. In our analysis we consider only the algorithm and the FPGA architecture. We have verified our analysis with Xilinx ISE synthesis tool [13] and found that for the Recursive Karatsuba multiplier our estimation of the LUTs used match the tool results. For the General Karatsuba multiplier our estimation for the LUTs was marginally lower than the results produced by the tool.

Recursive Karatsuba Multiplier : In an n bit Recursive Karatsuba Multiplier, each recursion reduces the size of the input by half while tripling the number of multiplications required. At each recursion, except the final, only XOR operations are involved. Let $m = 2^{(log_2 n)-k}$ be the size of the inputs, A and B, for the k^{th} recursion of an n bit multiplier. There are 3^k such m bit multipliers required. The A and B inputs are split into two: A_h, A_l and B_h, B_l respectively with each term having $m/2$ bits. $m/2$ two input XORs are required for the computation of $A_h + A_l$ and $B_h + B_l$ respectively (Equation :3). Each two input XOR requires one LUT on the FPGA. Combining the partial products as shown in the figure (1) is the last step of the recursion. Determining the output bits $m-2$ to $m/2$ and $3m/2-2$ to m requires $3(m/2-1)$ two input XORs each. The output bit $m-1$ requires 2 two input XORs. In all $(3m-4)$ two input $XORs$ are required to add the partial products. The number of LUTs required to combine the partial products is much lower. This is because each LUT implements a four input XOR. Each output bit $m/2$ to $3m/2-2$ requires one LUT, therefore $(m-1)$ LUTs are required for the purpose. In total, $2m-1$ LUTs are required for each recursion on the FPGA. The final recursion has $3^{(log_2 n)-1}$ two bit Karatsuba multipliers. The equation for the two bit Karatsuba multiplier is shown in equation (5).

$$\begin{aligned} C_0 &= A_0 B_0 \\ C_1 &= A_0 B_0 + A_1 B_1 + (A_0 + A_1)(B_0 + B_1) \\ C_2 &= A_1 B_1 \end{aligned} \quad (5)$$

This requires three LUTs on the FPGA; one for each of the output bits (C_0, C_1, C_2).

The total number of LUTs required for the n bit recursive Karatsuba multiplication is given by equation (6).

$$\begin{aligned} \#LUTS &= 3 * 3^{log_2 n-1} + \sum_{k=0}^{log_2 n-2} 3^k (2 * 2^{log_2 n-k} - 1) \\ &= \sum_{k=0}^{log_2 n-1} 3^k (2^{log_2 n-k+1} - 1) \end{aligned} \quad (6)$$

The FPGA usage for the Simple Karatsuba multiplier [11] is similar to that of the recursive Karatsuba multiplier.

Algorithm 1: gkmul *(General Karatsuba Multiplier)*

Input: A, B are multiplicands of n bits
Output: C of length $2n-1$ bits

/* **Define :** $M_x \rightarrow A_x B_x$ */
/* **Define :** $M_{(x,y)} \rightarrow (A_x + A_y)(B_x + B_y)$ */

begin
  for $i = 0$ *to* $n-2$ **do**
    $C_i = C_{2n-2-i} = 0$
    for $j = 0$ *to* $\lfloor i/2 \rfloor$ **do**
      if $i = 2j$ **then**
        $C_i = C_i + M_j$
        $C_{2n-2-i} = C_{2n-2-i} + M_{n-1-j}$
      else
        $C_i = C_i + M_j + M_{i-j} + M_{(j,i-j)}$
        $C_{2n-2-i} = C_{2n-2-i} + M_{n-1-j}$
          $+ M_{n-1-i+j} + M_{(n-1-j,n-1-i+j)}$
      end
    end
  end
  $C_{n-1} = 0$
  for $j = 0$ *to* $\lfloor (n-1)/2 \rfloor$ **do**
    if $n-1 = 2j$ **then**
      $C_{n-1} = C_{n-1} + M_j$
    else
      $C_{n-1} = C_{n-1} + M_j + M_{n-1-j} + M_{(j,n-1-j)}$
    end
  end
end

General Karatsuba Multiplier : The n bit General Karatsuba algorithm [11] is shown in Algorithm 1. The '+' in

the algorithm denotes XOR operation as the multiplication is in the field $GF(2^n)$. Each iteration of i computes two output bits C_i and C_{2n-2-i}. Computing the two output bits require same amount of resources on an FPGA. The line 6 in the algorithm is executed once for every even iteration of i, and is not executed for odd iterations of i. The term $M_j + M_{i-j} + M_{(j,i-j)}$ is computed with the four inputs A_j, A_{i-j}, B_j and B_{i-j}, therefore, on an FPGA, computing the term would require one LUT. For an odd i, the C_i would have $\lceil i/2 \rceil$ such LUTs whose outputs have to be added. The LUTs required for this can be obtained from equation (4).An even value of i would have an additional two inputs corresponding to $M_{i/2}$ which have to be added. The LUTs required for computing C_i $(0 \leq i \leq n-1)$ is given by equation (7).

$$\#LUT_{C_i} = \begin{cases} 1 & \text{if } i = 0 \\ \lceil i/2 \rceil + \#LUT(\lceil i/2 \rceil) & \text{if } i \text{ is odd} \\ i/2 + \#LUT(i/2+2) & \text{if } i \text{ is even} \end{cases} \quad (7)$$

The total number of LUTs required for the General Karatsuba multiplier is given by equation(8).

$$\#LUTS = 2\left(\sum_{i=0}^{n-2} LUT_{C_i}\right) + LUT_{C_{n-1}} \quad (8)$$

4 Hybrid Karatsuba Multiplier

In this section we present our Hybrid Karatsuba multiplier. We show how we combine techniques to maximize the utilization of each LUT resulting in minimum area. We also show that the proposed multiplier has minimum area delay product on the FPGA.

The table (1) compares the General and Simple Karatsuba algorithms for gate counts (two input XOR and AND gates), LUTs required on a Xilinx Virtex 4 FPGA and the percentage of LUTs under utilized. The percentage of under utilized LUTs is determined using the equation (9). LUT_k is the number of LUTs having k inputs.

$$\%UnderUtilizedLUTs = \frac{LUT_2 + LUT_3}{LUT_2 + LUT_3 + LUT_4} * 100 \quad (9)$$

n	General			Simple		
	Gates	LUTs	LUTs Under Utilized	Gates	LUTs	LUTs Under Utilized
2	7	3	66.6%	7	3	66.6%
4	37	11	45.5%	33	16	68.7%
8	169	53	20.7%	127	63	66.6%
16	721	188	17.0%	441	220	65.0%
29	2437	670	10.7%	1339	669	65.4%
32	2977	799	11.3%	1447	723	63.9%

Table 1: *Multiplication Comparison*

The Simple Karatsuba multiplier is not efficient for FPGA platforms as the number of under utilized LUTs is about 65%. For an n bit simple recursive multiplier the two bit multipliers take up approximately a third of the area (for $n = 256$). In a two bit multiplier, two out of three LUTs required, are under utilized (In equation (5), C_0 and C_2 result in under utilized LUTs). In addition to this, around half the LUTs used for each recursion is under utilized. The under utilized LUTs results in a bloated area requirement on the FPGA.

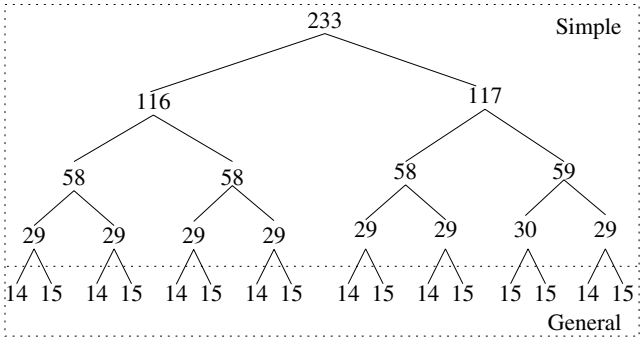

Figure 2. 233 Bit Hybrid Karatsuba Multiplier

The n-term General Karatsuba is more efficient on the FPGA for small values on n (Table 1) even though the gate count is significantly higher. This is because a large number of operations can be grouped in fours which fully utilizes the LUT. For small values of n ($n < 29$) the compactness obtained by the fully utilized LUTs is more prominent than the large gate count, resulting in low footprints on the FPGA. For $n \geq 29$, the gate count far exceeds the efficiency obtained by the fully utilized LUTs, resulting in larger footprints with respect to the Simple Karatsuba implementation.

In our proposed Hybrid Karatsuba multiplier, shown in the algorithm (2), the n bit multiplicands are split into two parts when the number of bits is greater than or equal to the threshold 29. The higher term has $\lfloor n/2 \rfloor$ bits while the lower term has $\lceil n/2 \rceil$ bits. If the number of bits of the multiplicand is less than 29 the General Karatsuba algorithm is invoked. The General Karatsuba algorithm ensures maximum utilization of the LUTs for the smaller bit multiplications, while the Simple Karatsuba algorithm ensures least gate count for the larger bit multiplications.

Algorithm 2: hmul *(Hybrid Karatsuba Multiplier)*

Input: The multiplicands A, B and their length n
Output: C of length $2n-1$ bits

1 **begin**
2 **if** $n < 29$ **then**
3 **return** $gkmul(A, B, n)$
4 **else**
5 $l = \lceil n/2 \rceil$
6 $A^{'} = A_{[n-1\cdots l]} + A_{[l-1\cdots 0]}$
7 $B^{'} = B_{[n-1\cdots l]} + B_{[l-1\cdots 0]}$
8 $C_{p1} = hmul(A_{[l-1\cdots 0]}, B_{[l-1\cdots 0]}, l)$
9 $C_{p2} = hmul(A^{'}, B^{'}, l)$
10 $C_{p3} = hmul(A_{[n-1\cdots l]}, B_{[n-1\cdots l]}, n-l)$
11 **return**
 $(C_{p3} << 2l) + (C_{p1} + C_{p2} + C_{p3}) << l + C_{p1}$
 ; /* $<<$ *indicates left shift* */
12
13 **end**
14 **end**

The i^{th} recursion $(0 \leq i < r)$ of the n bit multiplier has 3^i multiplications. The multipliers in this recursion have bit lengths $\lceil n/2^i \rceil$ and $\lfloor n/2^i \rfloor$. For simplicity we assume

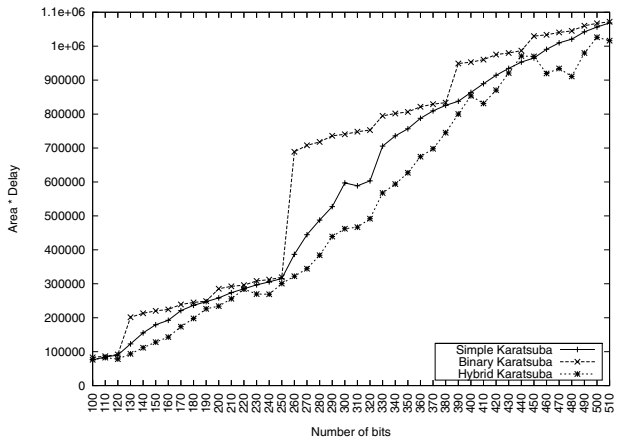

Figure 3. n bit multiplication vs Area X Delay

the number of gates required for the $\lceil n/2^i \rceil$ and $\lfloor n/2^i \rfloor$ bit multipliers is equal to that of a $\lceil n/2^i \rceil$ bit multiplier. The total number of AND gates required is the AND gates for the multiplier in the last recursion (i.e. $\lceil n/2^{r-1} \rceil$ bit multiplier) times the number of $\lceil n/2^{r-1} \rceil$ multipliers present.

$$\#AND = 3^{r-1}\lceil \frac{n}{2^r} \rceil \left(\lceil \frac{n}{2^{r-1}} \rceil + 1 \right) \tag{10}$$

The number of XOR gates required for the i^{th} recursion is given by $4\lceil \frac{n}{2^i} \rceil - 4$. The total number of two input $XORs$ is the sum of the $XORs$ required for last recursion, $\#XOR_{g_{r-1}}$, and the $XORs$ required for the other recursions, $\#XOR_{s_i}$.

$$\begin{aligned} \#XOR &= 3^{r-1}\#XOR_{g_{r-1}} + \sum_{i=0}^{r-2} 3^i \#XOR_{s_i} \\ &= 3^{r-1}\left(10\lceil \frac{n}{2^r} \rceil^2 - 7\lceil \frac{n}{2^r} \rceil + 1\right) + \sum_{i=0}^{r-2} 3^i \left(4\lceil \frac{n}{2^i} \rceil - 4\right) \end{aligned} \tag{11}$$

4.1 Performance Results

The graph in figure (3) compares the area delay product for the Hybrid Karatsuba multiplier with the Simple Karatsuba Multiplier and the Binary Karatsuba multipliers for increasing values of n. The area and delay was obtained by synthesizing each multiplier using Xilinx's ISE[13] for a Virtex 4 FPGA. The area was determined by the number of LUTs required for the multiplier. The delay is in nanoseconds. The graph shows that the area delay product for the Hybrid Karatsuba multiplier is lesser compared to the other multipliers.

5 DPA Resistant Multiplier

DPA attacks monitor the power consumption of the multiplier for several different inputs and then use statistical analysis to extract critical information about the key. One

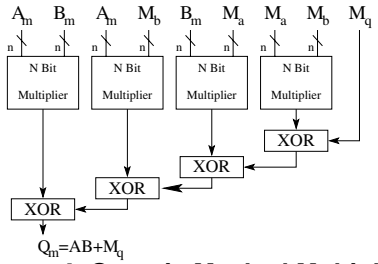

Figure 4. Generic Masked Multiplier

way to avoid DPA based attacks is to make the power consumption independent of the inputs. Masking the inputs with random values can achieve this[2]. However the masking should be in such a way that the required result can be inferred from the result obtained by the masked inputs.

For an n bit masked multiplier the inputs A and B are masked with random values M_a and M_b.

$$A_m = A + M_a \qquad B_m = B + M_b \tag{12}$$

The above equations can be rewritten as

$$A = A_m + M_a \qquad B = B_m + M_b \tag{13}$$

The generic way to build a masked multiplier is as follows. The masked inputs A_m, B_m and masks M_a, M_b are fed to the masked multiplier as shown in figure (4). The output of the masked multiplier is the sum of $(A * B)$ and another mask M_q as shown in the equation (14).

$$\begin{aligned} Q_m &= A_m B_m + A_m M_b + B_m M_a + M_a M_b + M_q \\ &= (A_m + M_a)(B_m + M_b) + M_q \\ &= AB + M_q \end{aligned} \tag{14}$$

The four N bit multipliers are implemented with any of the multiplication techniques described in the previous sections. Let MUL_A and MUL_X be the number of AND and XOR gates required for each multiplier respectively. The gates required by the masked multiplier is shown below. The $4(2n-1)$ extra $XORs$ are required to sum the outputs of the individual multiplications.

$$\begin{aligned} &\#AND : 4MUL_A(n) \\ &\#XOR : 4MUL_X(n) + 4(2n-1) \end{aligned} \tag{15}$$

Masking a multiplier requires more than four times the

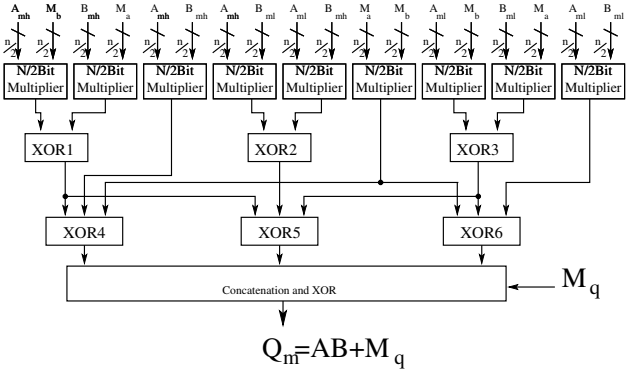

Figure 5. Proposed Masked Multiplier

gates as that of an unmasked multiplier. It is therefore important to search for masked techniques with the number of gates minimized. In this perspective we propose a masked multiplier (Figure 5) with lesser number of gates as compared with the generic masked multiplier. The multiplicands A and B of our masked multiplier are split into two terms just as in the Karatsuba algorithm. We then use an $n/2$ bit mask, M_a, to mask both the A_h and A_l terms. Similarly an $n/2$ bit mask, M_b, masks B_h and B_l as shown in equation (16).

$$\begin{aligned} A_{mh} &= A_h + M_a & A_{ml} &= A_l + M_a \\ B_{mh} &= B_h + M_b & B_{ml} &= B_l + M_b \end{aligned} \tag{16}$$

The masked inputs A_{mh}, A_{ml}, B_{mh}, B_{ml}, and the masks M_a, M_b and M_q are fed to the masked multiplier. The output (Equation 17) is the product of $(A * B)$ and the $2n-1$ bit mask M_q.

$$\begin{aligned} Q_m &= AB + M_q \\ &= [A_h x^{\frac{n}{2}} + A_l][B_h x^{\frac{n}{2}} + B_l] + M_q \\ &= [(A_{mh} + M_a)x^{\frac{n}{2}} + (A_{ml} + M_a)] \\ &\qquad [(B_{mh} + M_b)x^{\frac{n}{2}} + (B_{ml} + M_b)] + M_q \\ &= (A_{mh}B_{mh} + A_{mh}M_b + B_{mh}M_a + M_aM_b)x^n \\ &\quad + (A_{mh}B_{ml} + A_{mh}M_b + B_{ml}M_a \\ &\qquad + A_{ml}B_{mh} + A_{ml}M_b + B_{mh}M_a)x^{\frac{n}{2}} \\ &\quad + (A_{ml}B_{ml} + A_{ml}M_b + B_{ml}M_a + M_aM_b) + M_q \end{aligned} \tag{17}$$

The number of AND gates required is nine times that of an $n/2$ bit multiplier.

$$\#AND = 9MUL_A(n/2) \tag{18}$$

In addition to the $XORs$ required for the $n/2$ bit multipliers, XOR gates are required for combining the products, adding the mask M_q and for the gates $XOR1$ to $XOR6$. The $XOR1$, $XOR2$ and $XOR3$ require $(n-1)$ $XORs$ each. The $XOR4$, $XOR5$ and $XOR6$ require $2(n-1)$ two input $XORs$ each. Combining and adding the mask M_q requires $3(n-1)$ $XORs$.

$$\#XOR = 9MUL_X(n/2) + 12(n-1) \tag{19}$$

The graph in figure (6) shows the number of gates (AND + XOR) required verses the size of the multiplicands for the generic and proposed masked multipliers. The underlying multiplier in both cases is the Hybrid Karatsuba multiplier proposed earlier. From the graph the proposed masked multiplier has lesser gates requirements as compared to the generic masked multiplier and hence will require lesser area on the FPGA. The critical delay for $n = 233$ for the generic masked multiplier and the proposed masked multiplier was found to be almost the same (around 17 ns on a Xilinx Virtex 4 FPGA).

6 Conclusion

In this paper we proposed a hybrid technique of implementing the Karatsuba multiplier. Our proposed design results in best area $\times$ delay products on an FPGA compared to other Karatsuba implementations. We then proposed a new technique based on masking to prevent DPA attacks on the multiplier. Our technique is more efficient than conventional masking techniques. This multiplier forms an ideal base for a side channel resistant Elliptic Curve Crypto implementations on FPGA platforms.

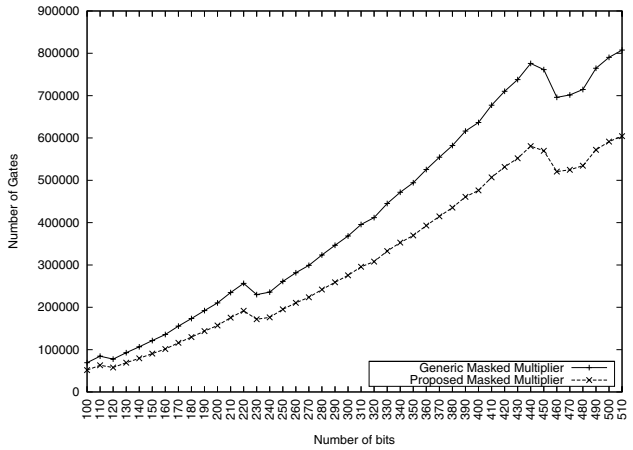

Figure 6. Gates required for n bit masked multiplication

References

[1] A. DeHon, *The Density Advantage of Configurable Computing*, IEEE Computer Society, 2000.

[2] J.D. Golic, *Techniques for Random Masking in Hardware*, IEEE Transactions on Circuits and Systems, vol 54, No 2, Pg 291-300, 2007.

[3] C. Grabbe, et. al., *FPGA Designs of Parallel High Performance $GF(2^{233})$ Multipliers*, International Symposium on Circuits and Systems, ISCS 2003.

[4] A. Karatsuba and Y. Ofman, *Multiplication of Multidigit Numbers on Automata*, Soviet Phys. Doklady (English Translation), vol 7, no 7, pg 595-596, 1963.

[5] P. Kocher, et. al., *Differential Power Analysis*, Advances in Cryptology-CRYPTO'99, 1999.

[6] F. R. Henriquez, et. al., *On Fully Parallel Karatsuba Multipliers for $GF(2^m)$*, Proceedings of the International Conference on Computer Science and Technology, CST 2003, 2003.

[7] S. Mangard, et. al., *Pinpointing the Side-Channel Leakage of Masked CMOS Implementations*, Workshop on Cryptographic Hardware and Embedded Systems 2006, CHES 2006.

[8] S.B. Ors, et. al., *Power-Analysis Attacks on an FPGA - First Experimental Results*, Workshop of Cryptographic Hardware and Embedded Systems - CHES 2003.

[9] S. Peter, et. al., *An Efficient Polynomial Multiplier in $GF(2^m)$ and its Applications to ECC Design*, Design, Automation and Test in Europe, DATE 2007, DATE.

[10] U.S. Department of Commerce/National Institute of Standards and Technology, *Digital Signature Standards*, Federal Information Processing Standards, Publication 186.

[11] A. Weimerskirch, et. al., *Generalizations of the Karatsuba Algorithm for Efficient Implementations*, http://eprint.iacr.org/2006/224.pdf, 2006.

[12] T. Wollenger. et. al., *Security on FPGAs: State of the Art Implementations and Attacks*, ACM Transactions in Embedded Computing Systems, Vol 3, No. 3, 2004.

[13] Xilinx, ISE Foundation, *http://www.xilinx.com/ise/logic_design_prod/foundation.htm*

[14] Xilinx, Virtex 4 User Guide, *http://direct.xilinx.com/bvdocs/userguides/ug070.pdf*