Scalable Computing: Practice and Experience
Volume 8, Number 4, pp. 411–422. http://www.scpe.org
ISSN 1895-1767
© 2007 SWPS

COMPLEXITY ANALYSIS FOR 4-INPUT/1-OUTPUT FPGAS APPLIED TO MULTIPLIER DESIGNS

NAZAR ABBAS SAQIB*

Abstract. Some algorithms are more efficient than others. The complexity of an algorithm is a function describing the efficiency of the algorithm which has two measures: *Space Complexity* and *Time Complexity*. In this paper, we present complexity analysis for FPGA based designs which is based on 4-input and 1-output LUT structure followed by the majority of FPGA manufacturers. The same procedure is then applied to Karatsuba-Offman Multiplier (KOM) because of two reasons. Firstly, due to the increased use of FPGAs especially for security applications, it seems logical to compare various architectures for their efficiencies in FPGAs. Secondly, for diverse security applications, it provides a prior estimation to hardware resources and achievable timing. We consider a 4-input and 1-output structure as a basic building block available in majority of FPGAs by different FPGA manufacturers. We then compare our theoretical and experimental results for KOM in FPGAs which are fairly convincible.

Key words. complexity analysis, field programmable gate arrays (FPGAs), Karatsuba-Ofman multiplier, cryptography, hardware implementations

1. Introduction. The use of internet for financial applications and electronic commerce has been tremendously increased which has made security a major concern. Public key cryptography [6] provides adequate security solution to those applications. First introduced in 1976, many algorithms were designed and implemented. The most popular schemes are due to RSA [31], ElGamal [9] and Elliptic Curve Cryptosystems (ECCs) [17, 23]. The security of these system is based on computational difficulty for solving some mathematical problems in modular arithmetic, multiplication being the most commonly used and costly operation.

Several quadratic and sub-quadratic space complexity multipliers have been reported in literature. Examples of quadratic multipliers can be found in [20, 18, 41, 42, 37, 13, 38, 35, 13, 28, 43, 11, 19, 32, 29, 30, 22, 7, 15]. On the other hand, some examples of sub-quadratic multipliers can be found in [24, 3, 25, 26, 12, 33, 36, 5, 10, 8, 40, 21]. The latter category offers low complexity especially for large values of n and therefore they are principally attractive for cryptographic applications.

The space and time complexities are the two measures for describing the efficiency of an algorithm. Space complexity is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm. In FPGAs, it refers to the hardware resources (configurable logic blocks, memory, etc) on the chip. Time complexity is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm. In FPGAs, it refers to all path delays including gate delays as well as routing overheads. A prior estimation of these two parameters has considerable importance for cost and speed estimations.

In VLSI designs, the estimation for both space and time complexities is relatively straightforward. If two pair of inputs $A\&B$ and $C\&D$ are XORed and their two outputs are ANDed, the space complexity is simply expressed as: #XORs = 2, #ANDs = 1. Similarly, if T_x is the delay for a single gate, time complexity for our example is $2T_x$, One T_x for XORing plus One T_x for ANDing. This is however not the case of an FPGA design. As the basic building block in majority of FPGAs has 4-inputs/1-output structure and also it acts like a Look Up Table (LUT), that is, the whole logic which bounds two, three or four inputs and produces one output, can be accommodated in just a single Look Up Table (LUT). Space complexity is therefore a single basic unit (a single LUT). In contrast to VLSI designs, Time complexity is not $1.T_x$ but it is $1.T_x$ plus path delays due to routing overheads in FPGAs. It has been observed that almost 70% of the total path delays is due to routing overheads in FPGAs. It is therefore difficult to link theoretical results to actual path delays in an FPGA based design. However certain optimizing techniques can be applied to reduce path delays by placing several registers at different stages of the design. At each move, the data travels from one stage to the next stage and hence the net path delay is the maximum delay between any two stages.

Recently, there is an emerging trend for implementing cryptographic primitives in hardware due to improved timing performances and also due to some security reasons. In contrast to software, hardware solutions offer high timing performances which is becoming critical at high speed links. On the other hand for security applications, it is more than that. The secret parameters (digital keys) in cryptographic primitives are stored in hardware

*Communication Systems Engineering (CSE) department, NUST Institute of Information Technology, National University of Sciences and Technology (NUST), Islamabad-Pakistan (nazar@niit.edu.pk)

and they are not easily accessible which enhances security. Another attractive features of FPGA based designs especially for security applications is due to ease in updating security algorithms as well as secret keys. The focus of this article is to devise a methodology for manipulating space and time complexities for various cryptographic primitives. We have selected Karatsuba-Ofman Multiplier (KOM) as our case study example.

The rest of this paper is organized as follows: Section 2 explains the procedure to perform complexity analysis in FPGAs. Section 3 demonstrate the same procedure for classical multipliers. In Section 4, Karatsuba-Ofman algorithm is explained for its space and time complexities in FPGAs. Section 5 shows the space-time benefits by combining the Karatsuba-Ofman multiplier and other multiplication schemes like classical multipliers. A comparison of all three multiplication schemes has been presented in Section 6. Conclusions are finally drawn in Section 7.

2. Complexity Analysis for FPGAs based Designs. FPGAs are being manufactured by many vendors like Xilinx [44], Altera [2], Atmel [4], Quick Logic [27], Actel [1], etc. All manufactures adopt different nomenclature for the hardware resources available on the chip. However the basic structure of almost all the FPGAs is the same. The basic building block in Xilinx FPGAs is called Configurable Logic Block (CLB). Each CLB has two slices and each slice contains one Look Up Table (LUT) other than additional logic. And each LUT has a 4-input and 1-output structure. Similarly, the basic building block in Altera FPGAs is called Logic Array Blocks (LAB). Each LAB contains ten logic elements (LEs) and each LE contains 4-input and 1-output LUT other than additional logic. However modern FPGAs even offer a 6-input and 1-output LUT [39]. Those building blocks are abundantly available in FPGAs. They can be configured into memory as well as into logic mode. Currently, FPGAs offer an integrated environment containing LUTs, Memory blocks, multipliers, transceivers, etc. In this article we focus on the smallest programmable unit in FPGAs, a LUT. We are considering FPGAs with 4-input and 1-output LUT structure for realizing complexity analysis. However the same procedure can be extended to advanced FPGAs with 6-input and 1-output LUTs.

First, in the context of 4-input and 1-output, we discuss two scenarios when number of inputs (IPs) are less than or equal to 4 and when they are greater than four.

When number of inputs ≤ 4 **:.** Let the output bit Z be the function of four input bits a, b, c, and d, then the significance of a LUT with 4-input and 1-output is that it would occupy just a single LUT in all the cases when Z is the function of two, three or four input bits. Also it does not matter what kind of Boolean logic is involved with those bits, that is,

- *When Z is the function of two bits i.e, Z= F(a,b)*
  Examples
  $$Z = a \oplus a.b;$$
  (One multiplication and one addition)

  or
  $$Z = a \oplus b \oplus a.b;$$
  (One multiplication and two additions)
- *When Z is the function of three bits i.e Z= F(a,b,c)*
  Examples
  $$Z = a \oplus b \oplus c \oplus a.b.c;$$
  (Two multiplications and three additions)

  or
  $$Z = a.b \oplus a.c \oplus b.c \oplus a.b.c;$$
  (Four multiplications and three additions)
- *When Z is the function of four bits i-e Z= F(a,b,c,d)*
  Examples
  $$Z = a.b \oplus a.c \oplus a.d \oplus b.c \oplus b.d \oplus a.b.c \oplus a.c.d \oplus b.c.d \oplus d;$$
  (Eleven multiplications and eight additions)

  or
  $$Z = a \oplus b \oplus c \oplus d \oplus a.b \oplus a.c \oplus a.d \oplus b.c \oplus b.d \oplus a.b.c \oplus b.c.d;$$
  (Nine multiplications and ten additions)

When number of inputs > 4 **:.** When Z is the function of more than four bits, it occupies more than one LUTs. For number of inputs from five to seven, Z utilizes two LUTs as four inputs go to the

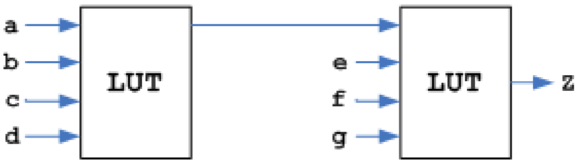

Fig. 2.1. *Seven input bits to occupy two LUTs*

1^{st} LUT and then its output is fed to the second one acting as an input for the 2^{nd} LUT as shown in Fig. 2.1.

As a rule of thumb, for Z as a function of k input bits, it uses some $k/3$ (nearest rounding) LUTs. e.g. The Z as a function of 10 and 11 inputs can be accommodated with $10/3 = 3.33 \cong 3$ and $11/3 = 3.66 \cong 4$ respectively.

The discussed results in this subsection can be applied to perform complexity analysis for any FPGAs based design. We apply this simple procedure to our two case studies for a classical multiplier and a Karatsuba-Ofman multiplier.

3. Complexity Analysis for a Classical Multiplier. We start with an example of a classical 4 × 4 bit multiplier as shown in Table 3.1.

Table 3.1
4-bit classical multiplier

			a_3	a_2	a_1	a_0
			b_3	b_2	b_1	b_0
			a_3b_0	a_2b_0	a_1b_0	a_0b_0
		a_3b_1	a_2b_1	a_1b_1	a_0b_1	
	a_3b_2	a_2b_2	a_1b_2	a_0b_2		
a_3b_3	a_2b_3	a_1b_3	a_0b_3			
z_6	z_5	z_4	z_3	z_2	z_1	z_0

From Table 3.1, one can quickly calculate the value of k and also the number of LUTs (dividing k by 3) for any z_j where j=0 to 6 as shown in Table 3.2.

Table 3.2
Complexity analysis for 4-bit classical multiplier

z_j	*FunctionF*	Partial Products	k_j	LUTs
z_0	$= F(a_0, b_0)$	$= a_0b_0$	2	1
z_1	$= F(a_0, b_0, a_1, b_1)$	$= a_1b_0 \oplus a_0b_1$	4	1
z_2	$= F(a_0, b_0, a_1, b_1, a_2, b_2)$	$= a_2b_0 \oplus a_1b_1 \oplus a_0b_2$	6	2
z_3	$= F(a_0, b_0, a_1, b_1, a_2, b_2, a_3, b_3)$	$= a_3b_0 \oplus a_2b_1 \oplus a_1b_2 \oplus a_0b_3$	8	3
z_4	$= F(a_1, b_1, a_2, b_2, a_3, b_3)$	$= a_3b_1 \oplus a_2b_2 \oplus a_1b_3$	6	2
z_5	$= F(a_2, b_2, a_3, b_3)$	$= a_3b_2 \oplus a_2b_3$	4	1
z_6	$= F(a_3, b_3)$	$= a_3b_3$	2	1
			Total	11

Hence, a 4-bit classical multiplier can be realized with no less than eleven 4-input and 1-output LUTs as shown in Fig. 3.1.

The procedure for performing complexity analysis of a 4-bit multiplier can be generalized to any n-bit multiplier which consists of three steps:

Step 1: Write down the number of inputs k_j for all partial sums z_j. It can be obtained first by writing n in the middle and then by writing all of its values from $(n-1)$ to 1 on its both sides. That gives the number of partial products for any partial sum z_j, that is,

$$1 \ldots\ldots \quad (n-2) \quad (n-1) \quad n \quad (n-1) \quad (n-2) \quad \ldots\ldots 1 \qquad (3.1)$$

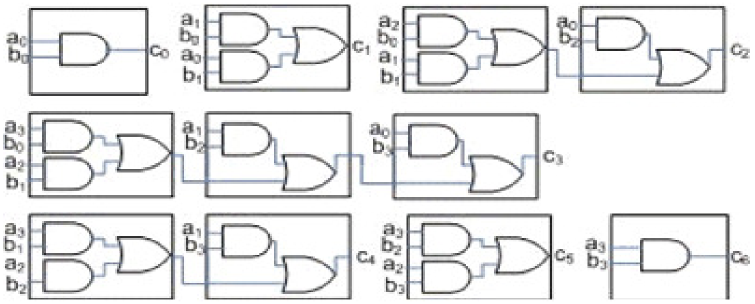

Fig. 3.1. *4-bit classical multiplier implementation using 4-input and 1-output LUTs*

For $n = 4$ (4-bit multiplier),

$$1 \quad 2 \quad 3 \quad 4 \quad 3 \quad 2 \quad 1 \tag{3.2}$$

As a single partial product contributes to two inputs, multiplying it by two, it give the number of inputs k_j for all partial sums z_j, that is,

$$2 \ldots\ldots \quad 2(n-2) \quad 2(n-1) \quad 2n \quad 2(n-1) \quad 2(n-2) \quad \ldots\ldots 2 \tag{3.3}$$

For $n = 4$ (4-bit multiplier),

$$2 \quad 4 \quad 6 \quad 8 \quad 6 \quad 4 \quad 2 \tag{3.4}$$

$$2n \quad 4(n-1) \quad 4(n-2) \quad \ldots\ldots 4 \tag{3.5}$$

Step 2: The number of LUTs for all partial sums z_j are manipulated by dividing each k_j by 3 and rounding it to the nearest integer value, that is,

$$\frac{2}{3} \quad \ldots. \quad \frac{2(n-2)}{3} \quad \frac{2(n-1)}{3} \quad \frac{2n}{3} \quad \frac{2(n-1)}{3} \quad \frac{2(n-2)}{3} \quad \ldots.. \quad \frac{2}{3} \tag{3.6}$$

For $n = 4$ (4-bit multiplier),

$$\frac{2}{3} \quad \frac{4}{3} \quad \frac{6}{3} \quad \frac{8}{3} \quad \frac{6}{3} \quad \frac{4}{3} \quad \frac{2}{3} \tag{3.7}$$

Step 3: The number of LUTs for all partial sums z_j are added to calculate total number of LUTs for any n-bit classical multiplier,

$$\frac{2}{3} + \cdots + \frac{2(n-2)}{3} + \frac{2(n-1)}{3} + \frac{2n}{3} + \frac{2(n-1)}{3} + \frac{2(n-2)}{3} + \ldots. + \frac{2}{3} \tag{3.8}$$

By simplifying,

$$\frac{2n}{3} + 2\frac{2(n-1)}{3} + 2\frac{2(n-2}{3} + \quad \ldots\ldots \quad + 2.\frac{2}{3} \tag{3.9}$$

$$\frac{2n}{3} + \frac{4}{3}\{(n-1) + (n-2) + \quad \ldots\ldots \quad + 1\} \tag{3.10}$$

The terms in brackets in Eq. 3.10 forms an arithmetic series for which the sum is equal to $\frac{n(n-1)}{2}$, by substituting:

$$\frac{2n}{3} + \frac{4}{3}\left\{\frac{n(n-1)}{2}\right\} = \frac{2}{3}n^2 \tag{3.11}$$

For 4-bit multiplier

$$\frac{2}{3} + \frac{4}{3} + \frac{6}{3} + \frac{8}{3} + \frac{6}{3} + \frac{4}{3} + \frac{2}{3} \tag{3.12}$$

By simplifying,

$$\frac{8}{3} + 2.\frac{6}{3} + 2.\frac{4}{3} + 2.\frac{2}{3} \tag{3.13}$$

$$3 + 4 + 2 + 2 = 11$$

By using this formula, one can calculate the gate complexity for any n-bit classical multiplier. Table 3.3 provides LUTs (cal) using the derived expression in Eq. 3.11 and also the number of LUTs (exp) experimented for first 40 values of n. The calculated LUTs exactly match with the experimental LUTs as we have instantiated LUT module implicitly in our VHDL code.

Table 3.3
Gate complexities for first 40 values of n using classical multiplier

n	LUTs (cal)	LUTs (Exp)	n	LUTs (cal)	LUTs (Exp)
1	1	1	21	294	294
2	3	3	22	323	323
3	6	6	23	353	353
4	11	11	24	384	384
5	17	17	25	417	417
6	24	24	26	451	451
7	33	33	27	486	486
8	43	43	28	523	523
9	54	54	29	561	561
10	67	67	30	600	600
11	81	81	31	641	641
12	96	96	32	683	683
13	113	113	33	726	726
14	131	131	34	771	771
15	150	150	35	817	817
16	171	171	36	864	864
17	193	193	37	913	913
18	216	216	38	963	963
19	241	241	39	1014	1014
20	267	267	40	1067	1067

4. Complexity Analysis for Karatsuba-Ofman Multiplier. Discovered in 1962, a divide-and-conquer algorithm due to Karatsuba and Ofman was the first algorithm [16] to accomplish polynomial multiplication in under $O(m^2)$ operations and reduces the complexity to $O(n^{log_2^3})$. Suppose that $n = 2l$ and $A = A^H 2^l + A^L$ and $B = B^H 2^l + B^L$ are $2l$-bit integers.
Then

$$\begin{aligned} AB &= (A^H 2^l + A^L)(B^H 2^l + B^L) \\ &= A^H B^H 2^{2l} + [(A^H + A^L)(B^H + B^L) - A^H B^H - A^L B^L]2^l + A^L B^L \end{aligned}$$

The product AB can be computed by performing three multiplications of l-bit integers along with two additions and two subtractions. More details about Karatsuba-Ofman multiplication can be seen in [14, 34].

Let we take again the example of a 4 × 4 multiplier using Karatsuba-Ofman multiplication scheme.

Let A and B the two multiplicands with,

$$A = a_3 a_2 a_1 a_0 \quad and \quad B = b_3 b_2 b_1 b_0 \tag{4.1}$$

Both A and B are divided into lower $A^L \& B^L$ and higher parts $A^H \& B^H$:

$$A^H = a_3 a_2 \quad and \quad B^H = b_3 b_2 \tag{4.2}$$

$$A^L = a_1 a_0 \quad and \quad B^L = b_1 b_0 \tag{4.3}$$

Then three multiplications are required to be performed:

1. First multiplication between A^H and B^H

$$A^H B^H = (a_3 a_2)(b_3 b_2) = H_2 H_1 H_0 \tag{4.4}$$

2. Second multiplication between A^L and B^L

$$A^L B^L = (a_1 a_0)(b_1 b_0) = L_2 L_1 L_0 \tag{4.5}$$

For third multiplication the higher and the lower parts of both the operands are XORed.

$$M^A = A^H \oplus A^L = (a_3 a_2) \oplus (a_1 a_0) = m_{a1} m_{a0} \tag{4.6}$$

$$M^B = B^H \oplus B^L = (b_3 b_2) \oplus (b_1 b_0) = m_{b1} m_{b0} \tag{4.7}$$

3. Third multiplication between M^A and M^B

$$M^A M^B = (m_{a1} m_{a0})(m_{b1} m_{b0}) = M_2 M_1 M_0 \tag{4.8}$$

Finally the overlapping of the three partial products is performed:

Table 4.1
Overlapping Function for a 4-bit Karatsuba-Ofman Multiplier

		H_2	H_1	H_0			
		L_2	L_1	L_0			$\oplus$
		M_2	M_1	M_0			$\oplus$
H_2	H_1	H_0		L_3	L_1	L_0	$\oplus$
z_6	z_5	z_4	z_3	z_2	z_1	z_0	

By looking at the above expressions one can estimate the resource utilization as follows:

1. Three $n/2$ multiplications are always performed by using Karatsuba-Ofman multiplication scheme. For a 4 × 4 Karatsuba-Ofman multiplier, it therefore requires three 2-bit multipliers as it is shown in Eqs. 4.4,4.5, and 4.8. A 2- bit multiplier using Karatsuba-Ofman multiplication scheme costs 3 LUTs, hence a total of 9 LUTs are being used.

2. For third multiplication the two inputs of the multiplier are to be XORed as it has been shown in Eqs. 4.6 and 4.7. They always require some $2 \times n/2$ XOR operations, and the same amount of LUTS i-e n LUTs. For $n = 4$, four LUTs are therefore utilized.
3. Finally the overlapping part is concluded with $3n - 4$ XORs thus consuming $(3n - 4)/3 = n - 1$ LUTs. For a 4-bit multiplier it is evident the utilization of three LUTs in obtaining z_3, z_4 and z_5, we call them as output XORs.

The total number of LUTs for a 4-bit Karatsuba-Ofman multiplier can be obtained by adding all LUTs from the above three steps which are 15. Some other results can also be deduced:

- LUTs due to input XORs $= 2(n/2) = n$
- LUTs due to output XORs $= n - 1$
- LUTs due to both input & output XORs $= n + (n - 1) = 2n - 1$
- LUTs due to three multipliers= 3× LUTs used by the base multiplier

The above procedure can be extended to generalize the expression for the estimation of number of LUTs for any n-bit Karatsuba-Ofman multiplier. We select a 4-bit Karatsuba-Ofman multiplier as a base multiplier, then,

For a 4-bit Karatsuba-Ofman multiplier $(n = 4)$:
Total number of LUTs $= 15$

For an 8-bit Karatsuba-Ofman multiplier $(n = 8)$:
Total number of LUTs

$$\begin{aligned}
&= \text{LUTs due to input/output XORs} + 3\times \text{ LUTs used by the 4-bit multiplier} \\
&= (2n - 1) + 14(3)^1 \\
&= 15 + 15(3)^1 = 15(3)^0 + 15(3)^1 = K_1
\end{aligned}$$

For a 16-bit Karatsuba-Ofman multiplier $(n = 16)$:
Total number of LUTs

$$\begin{aligned}
&= \text{LUTs due to input/output XORs} + 3\times \text{ LUTs used by the 8-bit multiplier} \\
&= (2n - 1) + 3 \times K_1 \\
&= 31 + 3\left\{15(3)^0 + 15(3)^1)\right\} = 31 + 15(3)^1 + 15(3)^2 = K_2
\end{aligned}$$

For a 32-bit Karatsuba-Ofman multiplier $(n = 32)$:
Total number of LUTs

$$\begin{aligned}
&= \text{LUTs due to input/output XORs} + 3\times \text{ LUTs used by the 16-bit multiplier} \\
&= (2n - 1) + 3 \times K_2 \\
&= 63 + 3\left\{31 + 15(3)^1 + 15(3)^2\right\} = 63 + 31(3)^1 + 15(3)^2 + 15(3)^3 = K_3
\end{aligned}$$

For a 64-bit Karatsuba-Ofman multiplier $(n = 64)$:
Total number of LUTs

$$\begin{aligned}
&= \text{LUTs due to input/output XORs} + 3\times \text{ LUTs used by the 32-bit multiplier} \\
&= (2n - 1) + 3 \times K_3 \\
&= 127 + 3\left\{63 + 31(3)^1 + 15(3)^2 + 15(3)^3\right\} \\
&= 127 + 63(3)^0 + 31(3)^2 + 15(3)^3 + 15(3)^4
\end{aligned}$$

On continuing in a similar way, we can generalize the above expressions for any n:

$$15(3)^k + \left\{\frac{2n-1}{1}3^0 + (\frac{2n}{2} - 1)3^1 + (\frac{n}{2} - 1)3^2 + (\frac{n}{4} - 1)3^3 + \cdots (\frac{n}{k-1} - 1)3^{k-1}\right\} \tag{4.9}$$

where k is the number of iterations and it is calculated as: $k = log_2(n) - 2$. The subtraction of factor of 2 is due to the selection of 4-bit multiplier as a base multiplier which removes two iterations for 2 and 4 bit multiplications.

Rewriting Eq. 4.9,

$$15(3)^k + \left\{\frac{2n}{1}3^0 + \frac{2n}{2}3^1 + \frac{n}{2}3^2 + \cdots + \frac{n}{k-1}3^{k-1}\right\} - \left\{3^0 + 3^1 + 3^2 + \cdots + 3^{k-1}\right\} \tag{4.10}$$

The terms in brackets in Eq. 4.10 form a geometric series similar to $a + ar + ar^2 + ar^3 + \cdots$ where 'a' represents the initial value and 'r' is the ratio which can be obtained by dividing a value to its previous one. The sum of n^{th} terms for that series can be calculated by the formula:

$$S_n = a(1 - r^n)/(1 - r) \tag{4.11}$$

The sum of n^{th} series for the two geometric expressions in Eq. 4.10 can be manipulated by using the formula in Eq. 4.11.

For the first series,

$$\left\{\frac{2n}{1}3^0 + \frac{2n}{2}3^1 + \frac{n}{2}3^2 + \cdots + \frac{n}{k-1}3^{k-1}\right\} \tag{4.12}$$

Initial value = $a = 2n$ & ratio = $r = 3/2$
Therefore the sum of n^{th} terms is:

$$= 4n\left[(3/2)^k - 1\right] \tag{4.13}$$

For the second series,

$$\left\{3^0 + 3^1 + 3^2 + \cdots + 3^{k-1}\right\} \tag{4.14}$$

Initial value = $a = 1$ & ratio=r= 3
Therefore the sum of the n^{th} terms is:

$$= 1/2\left[3^k - 1\right] \tag{4.15}$$

Substituting Eqs. 4.13 and 4.13 into Eq. 4.10,

$$15(3)^k + 4n\left[(3/2)^k - 1\right] - 1/2\left[(3)^k - 1\right] \tag{4.16}$$

Eq. 4.16 can be written in terms of just 'n' by substituting the value of 'k'

$$15(3)^{log_2(n)-2} + 4n\left[(3/2)^{log_2(n)-2} - 1\right] - 1/2\left[(3)^{log_2(n)-2} - 1\right] \tag{4.17}$$

where $k = log_2(n) - 2$

By using the formula in Eq. 4.17, we can calculate the space complexity for several $n = 2^k$-bit Karatsuba-Ofman multipliers as shown in Table 4.2. Table 4.2 also provides our experimental results for the same values which shows minor difference to the calculated values to non-optimal behavior of HDL (Hardware Description Language) compilers.

5. Complexity Analysis for Karatsuba-Ofman multiplier using Hybrid approach. In order to construct a bigger multiplier for any larger value 'm', we can use Karatsuba-Ofman multiplication approach by using a smaller multiplier recursively. The smaller multiplier represents the end point where recursion process exactly starts and it is termed as a base multiplier. A base multiplier can be constructed by any other multiplication approach like classical multiplication scheme as well. For example, we can construct an 8-bit multiplier from three 4-bit multipliers. Similarly a 16-bit multiplier can be constructed by using three 8-bit

Table 4.2
Space complexity for $n = 2^k$-bit Karatsuba-Ofman multiplier in terms of LUTs

n	LUTs (cal)	LUTs (Exp)
2	3	3
4	15	14
8	60	60
16	211	212
32	696	698
64	2215	2221
128	6900	6918
256	21211	21265
512	64656	64818

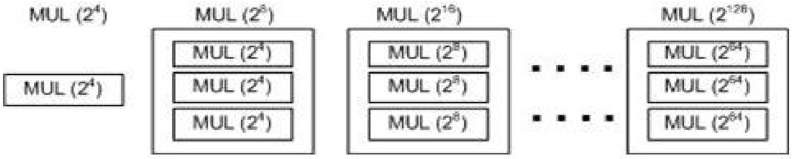

Fig. 5.1. *4-bit classical multiplier implementation using 4-input and 1-output LUTs*

mutipliers and so on. A block diagram representation of this hierarchical setup by selecting a 4-bit multiplier as a base multiplier is shown in Fig. 5.1.

Karatsuba-Ofman multiplier therefore can be viewed as a long array of base multipliers in middle and a logic mapping required for input and output (overlapping) XOR operations as it has been depicted in Fig. 5.2.

The selection of the base multiplier is therefore critical to save the hardware resources. The saving of few LUTs in the base multiplier helps in saving significant number of LUTs for large values of n. A hybrid approach is therefore used which dictates the use of other multiplication schemes along with Karatsuba-Ofman multiplication. We have implemented 4-bit Karatsuba-Ofman multiplier using the classical approach (school method) which seems to be economical as compared to 4-bit Karatsuba-Ofman multiplier as it occupies 11 LUTs instead of 15 LUTs. The change of only base multiplier does not require any change in the formula for complexity analysis, the factor of 15 is simply replaced with 11. The formula for an hybrid Karatsuba-Ofman multiplier using a 4-bit classical multiplier as a base multiplier is shown in Eq. 5.1.

$$11(3)^{log_2^n-2} + 4n\left[(3/2)^{log_2^n-2} - 1\right] - 1/2\left[3^{log_2^n-1} - 1\right] \tag{5.1}$$

By using Eq. 5.1, the space complexity for hybrid Karatsuba-Ofman multiplier can be manipulated as shown in Table 5.1.

Table 5.1
Space complexity for $n = 2^k$-bit Hybrid Karatsuba-Ofman multiplier in terms of LUTs

n	LUTs (cal)	LUTs (Exp)
2	3	3
4	11	11
8	48	45
16	175	168
32	588	567
64	1891	1828
128	5928	5739
256	18295	17728
512	55908	54207

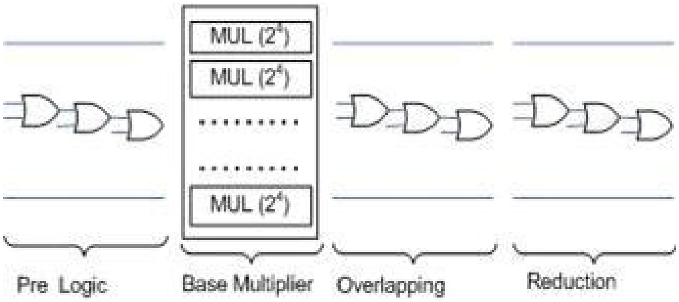

FIG. 5.2. *Flattend Image of Karatsuba-Ofman multiplier using a $MUL(2)^4$ as a base multiplier*

6. Performance Results. The achieved results for the space complexities of classical, Karatsuba-Ofman and hybrid Karatsuba-Ofman multiplication schemes can be combined for comparison purposes as shown in Table 6.1.

TABLE 6.1
Space complexity for $n = 2^k$-bit Classical, Karatsuba-Ofman, and Hybrid Karatsuba-Ofman multiplication schemes in terms of LUTs

n	LUTs (cal) Classical multiplier	LUTs (cal) Karatsuba-Ofman multiplier	LUTs (cal) H. Karatsuba-Ofman multiplier
2	3	3	3
4	11	15	11
8	43	60	48
16	171	211	175
32	683	696	588
64	2731	2215	1891
128	10923	6900	5928
256	43691	21211	18295
512	174763	64656	55908

It can be seen from Table 6.1 that classical multiplication schemes proves to be more economical for $n < 32$ when complexity analysis is performed for FPGAs based designs. For $n > 32$, however, hybrid Karatsuba-Ofman multiplication approach proves to be more economical.

7. Conclusion. In this paper, we explained in detail how to perform complexity analysis for an FPGA based design. We applied that procedure for manipulating space complexities for a classical Karatsuba-Ofman multiplier, Karatsuba-Ofman multiplier and an Hybrid Karatsuba-Ofman multiplier. It has been shown that obtained experimental results are exactly in match with those of theoretical manipulations in all three cases. The similar procedure can be extended to realize complexity analysis for other cryptographic primitives. The comparison tables for all three multiplication schemes can be utilized for selecting a base multiplier to construct a bigger multiplier as it is required in cryptographic applications. Our future work includes the construction of a low cost multiplier in FPGAs on the basis of the results obtained in this paper. Also we used a 4-input and 1-output structure for a LUT as the basic building block to perform complexity analysis for an FPGA based design. Modern FPGAs however offer a 6-input and 1-output structure for their basic building block. We have also planned to extend our manipulations for those FPGA devices.

REFERENCES

[1] ACTEL, 2008. Available at: `http://www.actel.com/`
[2] ALTERA, 2008. Available at: `http://www.xilinx.com/`
[3] C. P. AND, *A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields*, IEEE Transactions on Computers, 45(7) (1996), pp. 856–861.

[4] Atmel, 2008. Available at: `http://atmel.com/`

[5] J. C. Bajard, L. Imbert, and G. A. Jullien, *Parallel Montgomery Multiplication in GF(2^k) Using Trinomial Residue Arithmetic*, in 17th IEEE Symposium on Computer Arithmetic (ARITH-17 2005), 27-29 June 2005, Cape Cod, MA, USA, IEEE Computer Society, 2005, pp. 164–171.

[6] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory, IT-22 (1976), pp. 644–654.

[7] H. Fan and Y. Dai, *Fast Bit-Parallel GF(2^n) Multiplier for All Trinomials*, IEEE Trans. Computers, 54 (2005), pp. 485–490.

[8] H. Fan and M. A. Hasan, *A New Approach to Subquadratic Space Complexity Parallel Multipliers for Extended Binary Fields*. Centre for Applied Cryptographic Research (CACR) Technical Report CACR 2006-02, 2006. available at: `http://www.cacr.math.uwaterloo.ca/`

[9] T. E. Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, in Proceedings of CRYPTO 84 on Advances in cryptology, New York, NY, USA, 1985, Springer-Verlag New York, Inc., pp. 10–18.

[10] J. Gathen and J. Shokrollahi, *Efficient FPGA-Based Karatsuba Multipliers for Polynomials over F_2.*, in Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers, vol. 3897 of Lecture Notes in Computer Science, Springer-Verlag, 2006, pp. 359–369.

[11] D. Gollmann, *Equally Spaced Polynomials, Dual Bases, and Multiplication in F_{2^n}*, IEEE Trans. Computers, 51 (2002), pp. 588–591.

[12] C. Grabbe, M. B., J. Gathen, J. Shokrollahi, and J. Teich, *A High Performance VLIW Processor for Finite Field Arithmetic*, in 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), 22-26 April 2003, Nice, France, CD-ROM/Abstracts Proceedings, IEEE Computer Society, 2003, p. 189.

[13] A. Halbutogullari and Ç. K. Koç, *Parallel Multiplication in using Polynomial Residue Arithmetic*, Des. Codes Cryptography, 20 (2000), pp. 155–173.

[14] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[15] J. L. Imana, J. M. Sanchez, and F. Tirado, *Bit-Parallel Finite Field Multipliers for Irreducible Trinomials*, IEEE Transactions on Computers, 55 (2006), pp. 520–533.

[16] A. Karatsuba and Y. Ofman, *Multiplication of Multidigit Numbers on Automata*, Soviet Phys. Doklady (English Translation), 7 (1963), pp. 595–596.

[17] N. Koblitz, *CM-Curves with Good Cryptographic Properties.*, in CRYPTO, vol. 576 of Lecture Notes in Computer Science, Springer, 1991, pp. 279–287.

[18] Ç. K. Koç and T. Acar, *Montgomery Multiplication in GF(2^k).*, Designs, Codes and Cryptography, 14 (1998), pp. 57–69.

[19] S. O. Lee, S. W. Jung, C. H. Kim, J. Yoon, J. Y. Koh, and D. Kim, *Design of Bit Parallel Multiplier with Lower Time Complexity*, in Information Security and Cryptology - ICISC 2003, 6th International Conference, Seoul, Korea, November 27-28, 2003, Revised Papers, vol. 2971 of Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 127–139.

[20] E. D. Mastrovito, *VLSI Designs for Multiplication over Finite Fields f (2^m).*, in Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 6th International Conference, AAECC-6, Rome, Italy, July 4-8, 1988, Proceedings, vol. 357 of Lecture Notes in Computer Science, Springer-Verlag, 1989, pp. 297–309.

[21] P. L. Montgomery, *Five, Six, and Seven-Term Karatsuba-Like Formulae*, IEEE Trans. Comput., 54 (2005), pp. 362–369.

[22] C. Nègre, *Quadrinomial Modular Arithmetic using Modified Polynomial Basis*, in International Symposium on Information Technology: Coding and Computing (ITCC 2005), Volume 1, 4-6 April 2005, Las Vegas, Nevada, USA, IEEE Computer Society, 2005, pp. 550–555.

[23] N. I. of Standards and Technology, *Recomended Elliptic Curves for Federal Government Use*, 1997.

[24] C. Paar, *Efficient VLSI Architectures for Bit Parallel Computation in Galois Fields*, PhD thesis, Universität GH Essen, 1994.

[25] C. Paar, P. Fleischmann, and P. Roelse, *Efficient Multiplier Architectures for Galois Fields GF(2 ^{4n})*, IEEE Trans. Computers, 47 (1998), pp. 162–170.

[26] C. Paar, P. Fleischmann, and P. Soria-Rodriguez, *Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents*, IEEE Trans. Computers, 48 (1999), pp. 1025–1034.

[27] QuickLogic, 2008. Available at: http://quicklogic.com/.

[28] A. Reyhani-Masoleh and M. A. Hasan, *A New Construction of Massey-Omura Parallel Multiplier over f(2)*, IEEE Trans. Computers, 51 (2002), pp. 511–520.

[29] A. ReyhaniMasoleh and M. A. Hasan, *Efficient Multiplication Beyond Optimal Normal Bases*, IEEE Trans. Computers, 52 (2003), pp. 428–439.

[30] A. Reyhani-Masoleh and M. A. Hasan, *Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over GF($2^{\{}m\}$)*, IEEE Trans. Computers, 53 (2004), pp. 945–959.

[31] R. L. Rivest, A. Shamir, and L. M. Adelman, *A METHOD FOR OBTAINING DIGITAL SIGNATURES AND PUBLIC-KEY CRYPTOSYSTEMS*, Tech. Report MIT/LCS/TM-82, 1977.

[32] F. Rodríguez-Henríquez and Ç. K. Ko, *Parallel Multipliers Based on Special Irreducible Pentanomials*, IEEE Trans. Computers, 52 (2003), pp. 1535–1542.

[33] F. Rodríguez-Henríquez and Ç. K. Koç, *On Fully Parallel Karatsuba Multipliers for GF*(2^m), in International Conference on Computer Science and Technology (CST 2003), Cancun, Mexico, May 2003, pp. 405–410.

[34] F. Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pèrez, and C. K. Koc, *Cryptographic Algorithms on Reconfigurable Hardware (Signals and Communication Technology)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[35] E. Savas, A. F. Tenca, and Ç. K. Koç, *A Scalable and Unified Multiplier Architecture for Finite Fields GF() and GF(2^m)*, in Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings, vol. 1965 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 277–292.

[36] B. Sunar, *A Generalized Method for Constructing Subquadratic Complexity GF(2^k) Multipliers*, IEEE Trans. Computers, 53 (2004), pp. 1097–1105.

[37] B. Sunar and Ç. K. Koç, *Mastrovito Multiplier for All Trinomials*, IEEE Trans. Computers, 48 (1999), pp. 522–527.

[38] B. Sunar and Ç. K. Koç, *An Efficient Optimal Normal Basis Type II Multiplier*, IEEE Trans. Computers, 50 (2001), pp. 83–87.

[39] Virtex5, 2008. Available at: `http://www.xilinx.com/support/documentation/virtex-5.htm`

[40] A. Weimerskirch and C. Paar, *Generalizations of the Karatsuba Algorithm for Efficient Implementations.* Ruhr-Universität-Bochum, Germany. Technical Report, 2003. available at: `http://www.crypto.ruhr-uni-bochum.de/en_publications.html`

[41] H. Wu and M. A. Hasan, *Low Complexity Bit-Parallel Multipliers for a Class of Finite Fields*, IEEE Trans. Computers, 47 (1998), pp. 883–887.

[42] H. Wu, M. A. Hasan, and I. F. Blake, *New Low-Complexity Bit-Parallel Finite Field Multipliers Using Weakly Dual Bases*, IEEE Trans. Computers, 47 (1998), pp. 1223–1234.

[43] H. Wu, M. A. Hasan, I. F. Blake, and S. Gao, *Finite Field Multiplier Using Redundant Representation*, IEEE Trans. Computers, 51 (2002), pp. 1306–1316.

[44] Xilinx, 2008. Available at: `http://www.xilinx.com/`

Edited by: Francisco Rodriguez-Henriquez
Received: December 30th, 2007
Accepted: January 17th, 2008