A NOVEL DESIGN OF LARGE INTEGER NTT-BASEDMULTIPLIERS FOR FULLYHOMOMORPHIC ENCRYPTION

[1]G.Sreerama Yadav, PG Scholar, Gates Institute of Technology,Gooty

[2]Dr.K.C.Kullayappa,Associate Professor & HOD, Gates Institute of Technology,Gooty

ABSTRACT: ***Large integer multiplication has been widely used in fully homomorphic encryption (FHE). Implementing feasible large integer multiplication hardware is thus critical for accelerating the FHE evaluation process. In this paper, a novel and efficient operand reduction scheme is proposed to reduce the area requirement of radix-r butterfly units. We also extend the single port, merged-bank memory structure to the design of number theoretic transform (NTT) and inverse NTT (INTT) for further area minimization. In addition, an efficient memory addressing scheme is developed to support both NTT/INTT and resolving carries computations. Experimental results reveal that significant area reductions can be achieved for the targeted 786 432- and 1179 648-bit NTT-based multipliers designed using the proposed schemes in comparison with the related works. Moreover, the two multiplications can be accomplished in 0.196 and 2.21 ms, respectively, based on 90-nm CMOS technology. The low-complexity feature of the proposed large integer multiplier designs is thus obtained without sacrificing the time performance.***

1.INTRODUCTION

Completely homomorphic encryption (FHE) enables calculations to be done legitimately on ciphertexts for guaranteeing information protection on untrusted servers, in this manner pulling in much consideration for distributed computing applications. For the most part, FHE can be classified into three classifications: cross section based [1], integer based [2], and (ring) learning with blunders .

One of the primary difficulties in the improvement of commonsense FHE applications is to alleviate the amazingly high-computational multifaceted nature and asset requirements.For model, programming executions of FHE in elite PCs [4], [5] still

devour significant calculation time, especially for achieving enormous whole number increase which more often than not includes more than a huge number of bits. For grid based FHE, 785006bit augmentation is required for the little setting with a cross section measurement of 2048.

To quicken FHE activities, different efficient plans have been displayed in [12]–[17] to handle huge whole number increase dependent on the Schönhage–Strassen algorithm (SSA) [6]. The essential thought is to consolidate the parceled information of the BU to diminish the quantity of operands required in the summation task dependent on the natural properties of NTT.

Algorithm 1 : Schönhage–Strassen Algorithm (SSA)

// Inputs: X (multiplicand, u bits), Y (multiplier, u bits), B (base $= 2^b$)
// Output: $Z = X \times Y$
{
1. Initialization: Decompose X and Y, respectively, into a sequence of digits x_n and y_n, $0 \le n < M$, using base B. Then, assign $x_n = 0$ and $y_n = 0$ by zero padding for $M \le n < 2M$.
2. $X_k = \mathrm{NTT}(x_n)$, $Y_k = \mathrm{NTT}(y_n)$;
3. $Z_k = X_k \otimes Y_k$; // point-wise multiplication
4. $z_n = \mathrm{INTT}(Z_k)$;
5. $carry = 0$;
6. for $i = 0$ to $2M-1$ { // resolving carries computation
7. $sum = z_i + carry$;
8. $sb_i = sum \bmod B$; $carry = \lfloor sum/B \rfloor$;
9. }
10. return $Z = \sum_{i=0}^{2M-1} sb_i \times 2^{ib}$;
}

In addition, we extend [18] to the design of large integer multiplication using single-port, merged-bank (SPMB) memory for further reducing the memory area of the NTT implementation. A unified address generator unit (AGU) is also presented to support NTT/INTT (inverse NTT) and resolve carries computations. Experimental results show that applying the proposedschemes to the designsof786432- and1179648bit multipliers can achieve up to 52.34% and 18.73% area reductions compared to the related implementations in [16] and [15], respectively, and the area saving is obtained without compromising the time performance. The rest of this paper is organized as follows.

1.1FullyHomomorphicEncryption(FHE):

Homomorphic encryption is a form of encryption with an additional evaluation capability for computing over encrypted data without access to the secret key. The result of such a computation remains encrypted. Homomorphic encryption can be viewed as an extension of either symmetric-key or public-key cryptography. Homomorphic refers to homomorphism in algebra: the encryption and decryption functions can be thought as homomorphisms between plaintext and ciphertext spaces. Homomorphic encryption includes multiple types of encryption schemes that can perform different classes of computations over encrypted data.Some common types of

homomorphic encryption are partially homomorphic, somewhat homomorphic, leveled fully homomorphic, and fully homomorphic encryption. The computations are represented as either Boolean or arithmetic circuits. Partially homomorphic encryption encompasses schemes that support the evaluation of circuits consisting of only one type of gate, e.g., addition or multiplication.

2.LITERATURE SURVEY

C. Gentry, "*Fully homomorphic encryption using ideal lattices*,"[1] We propose a fully homomorphic encryption scheme – i.e., a scheme that allows one to evaluate circuits over encrypted data without being able to decrypt. Our solution comes in three steps. First, we provide a general result – that, to construct an encryption scheme that permits evaluation of arbitrary circuits, it suffices to construct an encryption scheme that can evaluate (slightly augmented versions of) its own decryption circuit; we call a scheme that can evaluate its (augmented) decryption circuit boots trappable. Next, we describe a public key encryption scheme using ideal lattices that is almost bootstrappable. Lattice-based cryptosystems typically have decryption algorithms with low circuit complexity, often dominated by an inner product computation that is in NC1. Also, ideal lattices provide both additive and multiplicative homomorphism's (modulo a public-key ideal in a polynomial ring that is represented as a lattice), as needed to evaluate general circuits. Unfortunately, our initial scheme is not quite bootstrappable – i.e., the depth that the scheme can correctly evaluate can be logarithmic in the lattice dimension, just like the depth of the decryption circuit, but the latter is greater than the former.

In the final step, we show how to modify the scheme to reduce the depth of the decryption circuit, and thereby obtain a boots trappable encryption scheme, without reducing the depth that the scheme can evaluate. Abstractly, we accomplish this by enabling the encrypted to start the decryption process, leaving less work for the decrypted, much like the server leaves less work for the decrypted in a server-aided cryptosystem.

We omit full details due to lack of space, but mention that one can construct an algorithm RandomizeCTE for our E2 that can be applied to ciphertexts output by EncryptE2 and EvaluateE2 , respectively, that induces equivalent output distributions. The circuit privacy of E2 immediately implies the

(leveled) circuit privacy of our (leveled) fully homomorphic encryption scheme. The idea is simple: to construct a random encryption ψ 0 of π from a particular encryption ψ of π, we simply add an encryption of 0 that has a much larger random "error" vector than ψ – super-polynomially larger, so that the new error vector statistically obliterates all information about ψ's error vector. This entails another re-definition of CE.

3.EXISTING METHOD

3.1. Introduction

In most digital signal processing (DSP) systems,a multiplier operation is one of the important hardware blocks. Some of the Signal Processing applications where a multiplier plays a vital role include digital filtering, digital communications and spectral analysis. Many current DSP applications are targeted at the portable battery operating systems, so as the primary design constraint the power dissipation become.

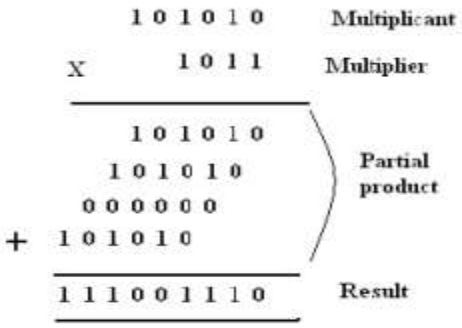
A multiplication is very expensive and slows the overall operation. Let us consider two unsigned binary numbers as X and Y and their bit length as M and N bits for them. For multiplication operation, it is useful to give X and Y values in the binary format.

$$X = \Sigma\, Xi\ 2^i \quad i = 0 \text{ to } M$$
$$Y = \Sigma\, Yj\ 2^j \quad j = 0 \text{ to } N$$

$$Z = X \times Y = \Sigma Z_k 2^k \quad k = 0 \text{ to } M+N-1 \tag{5.3}$$
$$= (\Sigma\, Xi\, 2^i \; i = 0 \text{ to } M)\,(\Sigma\, Yj\, 2^j \; j = 0 \text{ to } N) \tag{5.4}$$
$$= \Sigma\,(\Sigma\, X^i Y^j 2^{i+j})\; i = 0 \text{ to } M\text{-}1,\ j = 0 \text{ to } N\text{-}1 \tag{5.5}$$

The use of a single two input adder is a simplest way to perform a multiplication.The multiplication tasks M cycles for inputs that are M and N bits wide. This shift and addition of PP algorithm for multiplication adds together M partial products. It is very easy method to perform binary multiplication between them than decimal multiplication. In the value of each digit of a binary number can only be either 0 or 1 that is depending on the multiplier bit length, If the value is 0 the partial products can take as the copy of bits of the multiplicand. In digital logic circuits, this is simply an AND logic gate functionality. One off the speed multiplication operation is to resort to an approach similar to the manually computing of a multiplication process. The entire partial product are generated at the same time and organized in an array. A multi-operand addition is applied to compute the final product.

The process is shown in the below figure. This is set of operations that can be able map directly into hardware design then resulting structure is called array multiplier.



Example of manual multiplication

we came to learn that the new improved 14 Transistor having full adder blocks shows better result in the Threshold loss problem, power omission and speediness by designing using MOS transistor count.

In this approach, the given four important types of multiplier operations are there those are Array, Baugh wooly, Braun and Wallace tree, all of these are implemented using different types of adder blocks presented in Shalem.R in 199, then after we find out the better one in the operation performance like power, speed and area.

truncated multiplier

Truncated multiplication is one of the methods where only the most significant columns of the multiplication operation matrix are using and therefore the area required can be decreased. In Truncation multiplication method where the least significant columns are in the partial product matrix operation are not formed at any cast. The approach behind truncated multiplication operation is the same as when we deal with non-truncated multiplication operation than of the truncation degree operation approach. The process is shown in below Figure 2, where a truncation degree having of T = *3*, is applied as shown.

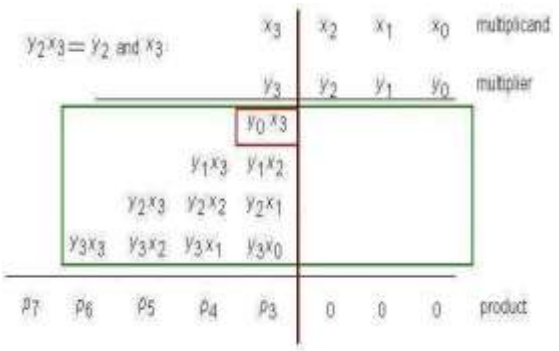


Fig 3.1:4x4 bit Binary Multiplication with truncation degree T=3

In the truncated multiplier the removal of unnecessary PPBs is composed of three processes:

1. Deletion,
2. Truncation
3. Rounding.

Deletion:

In truncated multiplier operation we start the multiplication operation process

with particular approach only.**Truncation Operation:**

Truncation Operation is a approach where the least significant columns are not formed which are in the Partial Products. The number of columns is not formed in this approach, where the term T is defines the degree of truncation operation and the T Least Significant Bits (LSB) of the product is always having the results in zero state.

Rounding:

Generally an n bit width of multiplicand and an n bit width of multiplier are form a 2^nbit width of product. In Some approaches the n bit output is desired to reduce the number of stored bits in the operation result

Let consider a 5x5 bit multiplier operation approach

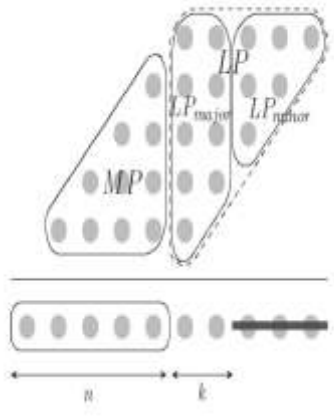


Fig 3.2:Partial-products matrix of a 5x5 bit multiplier

Advantages

- Memory occupation in the truncated multiplier is low compare to other multipliers.
- Delay will be changes with respect to the number of bits taken in the multiplier (4-bit,8-bit,12-bits) that is low.
- Speed also high i.eIf the delay is low then speed will be increase, those are inversely proposal to each other.

Disadvantages

- The main disadvantage of the truncated multiplier is it gives rounded value but not exactvalue.

4.PROPOSED WORK

4.1. Background and Notation

4.1.1.Schönhage–Strassen algorithm

The SSA [6] is an NTT-based solution for carrying out large integer multiplication. As shown in the SSA algorithm, let X and Y represent the u-bit integer multiplicand and multiplier, respectively,and the two operands arepartitionedinto M digits using base B, whereM = u/b and B = 2b. Moreover, zero padding is needed to extend both operands to 2M digits before performing cyclic convolution. The convolution is accomplished by doing NTT, point-wise multiplication, and INTT

sequentially. The final result Z is then obtained by resolving carries on the INTT output according to the base B. NTT can be viewed as a discrete Fourier transform defined over finitefieldZp [10]. An N-point NTT is defined as

$$X_k = \sum_{n=0}^{N-1} x_n (W_N)^{nk} \bmod p \qquad (1)$$

where 0≤k < N−1 andWN is a primitive Nth root of unity in Zp. Similarly, the INTT can be expressed as

$$x_k = N^{-1} \sum_{n=0}^{N-1} X_n (W_N)^{-nk} \bmod p. \qquad (2)$$

The prime p should be chosen such that N/2×(B−1)2 < p to avoid data overflow while performing cyclic convolution. A fast modulo-p operation can be obtained by choosing p as a special prime such as 264−232+1 [11]. As stated in [16], for an N-point NTT of small sizes (N ≤ 64), the value of WN is simply a power of 2, i.e., 2t with t = 192/N. Thus, a small-size NTT can be computed by using shift and modular additionoperationsratherthanmodularmultiplication,and can also be treated as a radix-r BU, written as

$$X_k = \sum_{n=0}^{r-1} x_n 2^{t \cdot nk \bmod 192} \bmod p. \qquad (3)$$

For large-size NTTs, the result can be obtained by successive use of even and odd point decompositions such as the wellknownCooley–Tukey algorithm and the BU unit. From (3), by taking advantage of 2192 mod p = 1, each 64-bit input of a BU is extended to 192 bits to relax the hardware required for carrying out the modulo-p operation. Let z be a 192-bit number. The z modulo-p operation can then be simplified as

$$z \equiv 2160a+2128b+296c+264d+232e+ f \equiv (232e+f)+(232d+a)−(232b+c)−(232a+d) \qquad (4)$$

where a, b, c, d, e, andf are 32-bit numbers. As a result, only modular additions and subtractions are needed to fulfill a fast modulo-p operation.

4.1.2. Merged-Bank Memory Addressing Scheme

Generally, compared to the multiport memory structures, single-port memory is preferred due to its area efficiency, but it might suffer from the limited bandwidth. Partitioning a single-port memory into multiple banks together with an efficient memory management scheme is usually adopted to increase the equivalent bandwidth. In [18], an area-efficient (AE) algorithm was developed by employing SPMB memory to further reduce the memory area in existing memory-based architectures. The basic idea is to divide the

merged-bank memoryinto two banks, which are read or written accordingto certain rules during operations. These rules ensure that when one of the banks is being read, the other is being written in every cycle.

For an N-point FFT implemented with a radix-r BU in the SPMB architecture, let n =[nm−1nm−2,...,n1 n0]2.The input data xn are loaded into memory according to the following rules:

Memory bank: nm−s−1

⊕nm−s−2 ⊕...⊕n0

Memory address: [nm−s−1,...,n1]2

(5)

where the symbol ⊕ denotes the exclusive-OR operation, m =log2 N and s =log2r. Thus, there are $[nm-1,...,nm-s]_2$

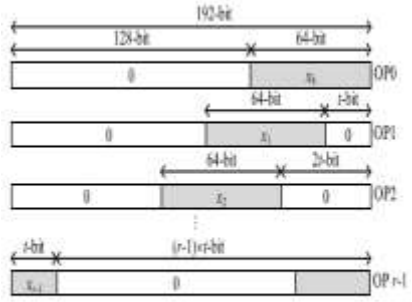


Fig. 1. Number of operands for X_1 (k = 1) using a radix-r BU.

Operands stored at a given address. A butterfly counter (BC) along with the gray code representation of the index is used to generate addresses for the BU in order to achieve conflict free memory access. From the following equation, the desired memory bank (BN) and memory address (MA) for each stage v can be obtained using the right rotation (RR) property, where ^ and stand for the unary reduction exclusive-OR operation and floor function, respectively:

$$\begin{aligned} \mathrm{BN}(\mathrm{BC}, v) &= {}^{\wedge}\mathrm{RR}(\mathrm{BC}, s \times v) \\ \mathrm{MA}(\mathrm{BC}, v) &= \lfloor RR(BC, s \times v)/2 \rfloor. \end{aligned} \quad (6)$$

4.2. Proposed low-cost multiplication design

4.2.1.Operand reduction scheme for butterfly unit

As can be observed from(3), the summation of the 192-bit operands x n s multiplied by the associated twiddle factors for a radix r BU can be interpreted as the summation of the left circular shift of xn by n ·k ·t (mod 192) bits. For instance, Fig. 1 illustrates the relationship among the left circular shift of xn s for computing X1 by a radix-r BU. Given the r operands (OPs), a direct implementation of the summation operation using a carry-save adder (CSA) tree will take at least log1.5 (r/2) levels, which leads to a large area requirement for large r. The result is then reduced to 64 bits by performing the 192-bit modulo-p operation.

Since there are 128 bits of zeros in each operand, we can take advantage of the zero elements by merging compatible

operands to reduce the number of effective operands. This in turn might reduce the level of the CSA tree. In this paper, the two operands OPi and OPj are said to be compatible if the positions of their constituent data points xi and xj do not overlap, such as OP2 and OPr−1 in Fig. 1, and they can be combined to yield a new operand. Note that for each Xk, $0 \le k \le r-1$, in (3), the set of its corresponding operands derived from the left circular shift of xn is determined by the indices n and k.

Given a radix-r BU, this paperexplorestheinherentfeatures in the set of operands for each Xk and proposes an efficient operand reduction algorithm to minimize the number of effective operands. Moreover, to increase the chance of merging operands, we extended the concept of compatible operands to consider the segmented data point xn described below. Applying the proposed schemes can reduce the number of operands by at least r/2 in each summation operation of Xk, $1 \le k \le r-1$, and retain r reduced-length operands for X0.

As a result, except for the X0 output, the lower bound of the level of required CSA trees becomes log1.5(r/4) , instead of log1.5(r/2) stated above, for the outputs of a radix-r BU. Let each NTT input data xnin(3) be partitioned into [64/t] words, where t =192/r. Then, xncanbe expressed as

$$x_n = \sum_{w=0}^{e-1} x_{n,w} 2^{tw} \qquad (7)$$

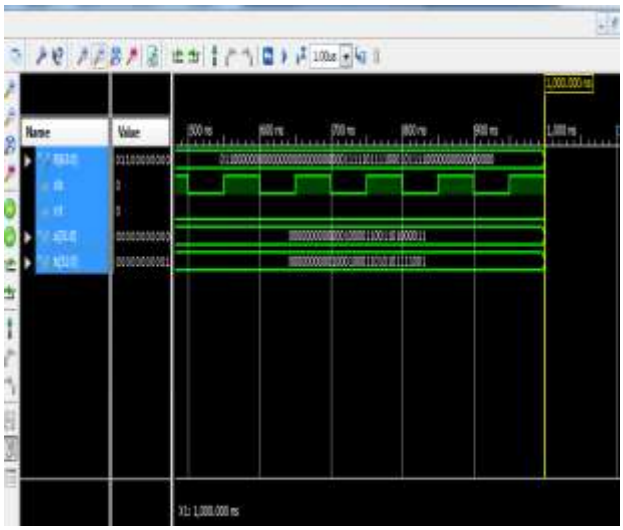
Where w denotes the wth word and e=64/t . The notation xn,w represents the wth word of the nth input data xn. Substituting (7) into (3), we obtain

$$X_k = \sum_{n=0}^{r-1} \left(\sum_{w=0}^{e-1} x_{n,w} 2^{tw} \right) \times 2^{t \cdot nk \bmod 192} \bmod p. \qquad (8)$$

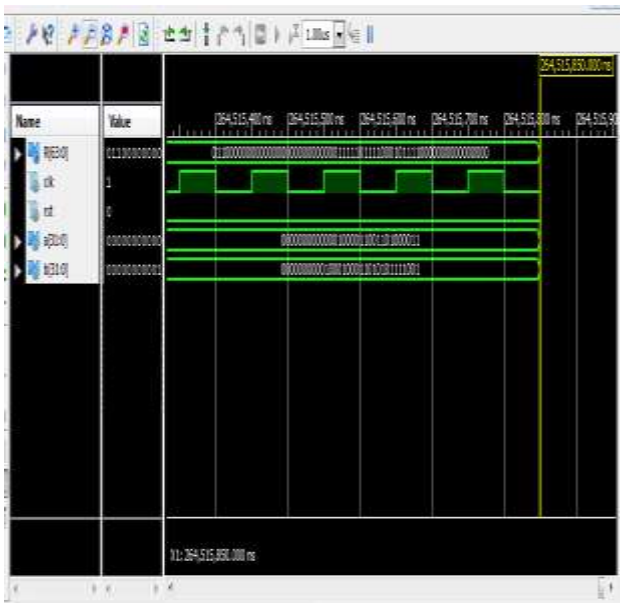
The benefit of adopting xn,w as a basic element in the NTT computation using radix-r BUs is to increase the freedom of choosing compatible operands, thus leading to an optimized solution of operand reduction. For example, OP0 and OP1, consisting of overlapped x0 and x1, respectively, in Fig. 1 are not compatible, but x0,0 and x1,0 are compatible if we treat the segmented xn,w as a new operand and assume that the value of e is large enough to decouple the overlap. Note that the segmentation of xn and merging of xn,w can be easily accomplished in hardware design with almost no hardware overhead. In the following, we show how to carry out the operation reduction process in a systematic and efficient way. Since the value of r is

basically a power of two for a radix-r BU, each input data xn,w in (8) will be circularly shifted left by (n -k+w)·t (mod 192) bits before being accumulated to form Xk. Let the period of (n·k+w)·t (mod 192) be defined as the absolute difference of the two closest values of n that lead to the same evaluated result for a given k. By carefully analyzing the period of (n·k+w)·t (mod 192) for different values of k, we have the following observations. Recall that each 64-bit xn is partitioned into e words, i.e., xn,w for 0 ≤ w<e, with each word consisting of t bits.

5.RESULTS



1.NTT-based multiplier designs can provide a significant area improvement in comparision with related works without compromising the time performance.



2.FHE, which demands low-complexity and high-speed large integer multiplication.

```
No Partitions were found in this design.

--------------------------


==========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 81    |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design
```

3.Timing Report of Large Multiplier clock based data

```
Data Path: m6/m1/count_0_1 to m6/m8/mul_11
                          Gate     Net
  Cell:in->out      fanout  Delay   Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   FDRSE:C->Q             2   0.514   0.532  m6/m1/count_0_1 (m6/m1/count_0_1)
   LUT2:I0->O             8   0.612   0.663  m6/i3<1>1 (m5/i3<1>)
   MUXF5:S->O             1   0.641   0.000  m5/m5/Mmux__COND_4_4_f5 (m5/m5/Mmux__COND_4_4_f5)
   MUXF6:I1->O            1   0.451   0.000  m5/m5/Mmux__COND_4_3_f6 (m5/m5/Mmux__COND_4_3_f6)
   MUXF7:I1->O           19   0.451   0.991  m5/m5/Mmux__COND_4_2_f7 (m5/k1<2>)
   LUT2:I1->O             1   0.612   0.360  m5/m8/m1/m5/Mxor_s_xo<0>_SW0 (N186)
   LUT4_D:I3->LO          1   0.612   0.130  m5/m8/m1/m5/Mxor_s_xo<0> (N689)
   LUT3:I2->O             3   0.612   0.454  m5/m8/m1/m10/Mxor_s_xo<0>11 (m5/m8/m1/N11)
   LUT4_D:I3->O           3   0.612   0.454  m5/m8/m1/m12/Mxor_s_xo<0>19 (m5/m8/m1/N01)
   LUT4:I3->O             0   0.612   0.000  m5/m8/m1/m12/cout (m5/m8/c1<7>)
   MUXCY:DI->O            1   0.773   0.000  m5/m8/Madd_AUX_6_addsub0001_cy<3> (m5/m8/Madd_AUX
   MUXCY:CI->O            1   0.052   0.000  m5/m8/Madd_AUX_6_addsub0001_cy<4> (m5/m8/Madd_AUX
   MUXCY:CI->O            1   0.052   0.000  m5/m8/Madd_AUX_6_addsub0001_cy<5> (m5/m8/Madd_AUX
   XORCY:CI->O            1   0.699   0.509  m5/m8/Madd_AUX_6_addsub0001_xor<6> (m5/m8/Madd_A
   LUT1:I0->O             1   0.612   0.000  m5/m8/Madd__AUX_6_Madd_cy<6>_rt (m5/m8/Madd__AUX_
   MUXCY:S->O             0   0.404   0.000  m5/m8/Madd__AUX_6_Madd_cy<6> (m5/m8/Madd__AUX_6_M
   XORCY:CI->O            1   0.699   0.000  m5/m8/Madd__AUX_6_Madd_xor<7> (m5/m8/o<11>)
   FDR:D                      0.268          m5/m8/mul_11
  ----------------------------------------
  Total                     13.361ns (9.287ns logic, 4.074ns route)
                                     (69.5% logic, 30.5% route)
```

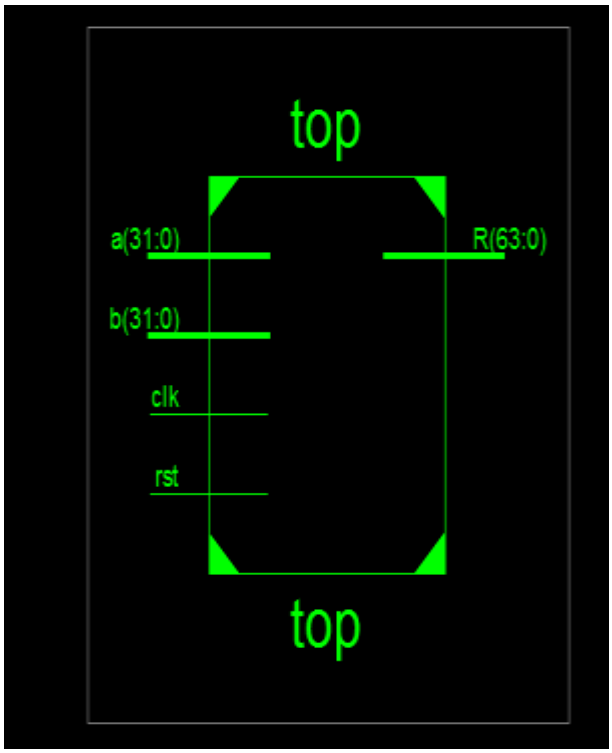
4.Synthesis Report for Large multipliers including Logic Utilization and Total Gate Delay 13.36ns

```
   LUT2:I0->O            1   0.612   0.000  Madd_R_lut<48> (Madd_R_lut<48>)
   MUXCY:S->O            1   0.404   0.000  Madd_R_cy<48> (Madd_R_cy<48>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<49> (Madd_R_cy<49>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<50> (Madd_R_cy<50>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<51> (Madd_R_cy<51>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<52> (Madd_R_cy<52>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<53> (Madd_R_cy<53>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<54> (Madd_R_cy<54>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<55> (Madd_R_cy<55>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<56> (Madd_R_cy<56>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<57> (Madd_R_cy<57>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<58> (Madd_R_cy<58>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<59> (Madd_R_cy<59>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<60> (Madd_R_cy<60>)
   MUXCY:CI->O           1   0.051   0.000  Madd_R_cy<61> (Madd_R_cy<61>)
   MUXCY:CI->O           0   0.051   0.000  Madd_R_cy<62> (Madd_R_cy<62>)
   XORCY:CI->O           1   0.699   0.357  Madd_R_xor<63> (R_63_OBUF)
   OBUF:I->O                 3.169          R_63_OBUF (R<63>)
  ----------------------------------------
  Total                    20.519ns (14.275ns logic, 6.244ns route)
                                    (69.6% logic, 30.4% route)

=========================================================================


Total REAL time to Xst completion: 18.00 secs
Total CPU time to Xst completion: 17.75 secs
```

5.Large integer Multipliers Lookup tables with route Map and Propagation Delay

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	649	4656	13%
Number of Slice Flip Flops	81	9312	0%
Number of 4 input LUTs	1203	9312	12%
Number of bonded IOBs	130	232	56%
Number of BUFGs	1	24	4%

6.Device utilization Table



7.Schematic circuit for NTT Multiplier

FUTURE SCOPE

An optimized FPGA implementation of a novel, fast and highly parallelized NTT-based polynomial multiplier architecture, which proves to be effective as an accelerator for lattice-based homomorphic cryptographic schemes.

As input-output (I/O) operations are as time-consuming as NTT operations during homomorphic computations in a host processor/accelerator setting, instead of achieving the fastest NTT implementation possible on the target FPGA, we focus on a balanced time performance between the NTT and I/O operations. Even with this goal, we achieved the fastest NTT implementation in literature, to the best of our knowledge

CONCLUSION

In this paper, we investigated plausible answers for accomplishing low-multifaceted nature NTT-based augmentation of enormous whole numbers for FHE, concentrating on the equipment execution. Initially, we have proposed a deliberate method for lessening the quantity of operands required for completing the radix-r butterfly calculation, a fundamental task in FFT/NTT applications.. Applying the proposed plans to the focused on 786432-piece and 1179648-piece NTT-based multiplier structures can give a significant zone improvement in correlation with related works without bargaining the time execution.

REFERENCES

[1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in Proc. 41st Annu. ACM Symp.Theory Comput. 2009, pp. 169–178.

[2] M. V. Dijk, C. Gentry, S. Halevl, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in Proc. Annu.Int.Conf. Theory Appl. Cryptograph.Techn., 2010, pp. 24–43. [3] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," ACM Trans. Comput. Theory, vol. 6, no. 3, 2012, Art.no. 13.[4] C. Gentry and S. Halevi, "Implementating gentry's fully-homomorphic encryption scheme," in Proc. Annu. Int. Conf. Theory Appl. Cryptograph.Techn., 2011, pp. 129–148.

[5] J.-S. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic

encryption over the integers," in Proc. Annu.Int. Conf. Theory Appl. Cryptograph.Techn., 2012, pp. 446–464.

[6] A. Schönhage and V. Strassen, "SchnellemultiplikationgroßerZahlen," Computing, vol. 7, nos. 3–4, pp. 281–292, 1971.