IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 8, Issue 6, Ver. I (Nov. - Dec. 2018), PP 34-38
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Implementation of Modular Reduction and Modular Multiplication Algorithms

Mishal Jasmine Ferrao[1], Mr. Kiran Kumar. V. G[2], Mrs. Megha N[3]
1(M.Tech. in VLSI Design and Embedded Systems, Sahyadri College of Engineering and Management, Adyar, Mangaluru, India)
2(Associate Professor, Dept. of Electronics and Communication Engineering, Sahyadri College of Engineering and Management, Adyar, Mangaluru, India)
3(Assistant Professor, Dept. of Electronics and Communication Engineering, Sahyadri College of Engineering and Management, Adyar, Mangaluru, India)
Corresponding Author: Mishal Jasmine Ferrao

Abstract: *Secure and reliable system has become an important exaction in the modern day offices. With the evolution of office equipment's from handwritten notes to desktops to digital storage of confidential information regarding business, client and employee information has to be stored in a secure manner. Cryptography, process of making the information classified, switch it to a form that may be non-readable, keeping it safe from unauthorized people uses modular operations in many of its algorithms. Unlike other arithmetic operations the length of the output does not vary in modular operations. The length and the value of the output is within the prime number used in the modular operation also called as modulo. In this paper, three modular reduction algorithms and one modular multiplication algorithm is implemented. The algorithms are implemented in Xilinx 14.2 using Spartan 6 as the device. The results are tabulated in this paper.*

Keywords: *Barrett reduction, Mod without Division. Montgomery Reduction, Montgomery Multiplication.*

Date of Submission: 22-12-2018 Date of acceptance: 07-01-2019

I. Introduction

Arithmetic operations like addition, multiplication are commonly used in many of the data processing applications. An operation like division is less frequently used but it is still important. There are a few applications that require only the remainder from the division process. The aim of the application might be to find the last two digits of a number after multiplication or exponentiation operation or find if a particular number is divisible by a particular number. For example, the last two digits of 99^{99} or find if 187596 is divisible by 82 and so on. There are also applications like cryptography where the inputs are to be wrapped around a particular input which is the key. In these instances modular arithmetic operations are employed. The other significant applications of modular operations are hashing, generation of random numbers, International Standard Book Number (ISBN) which is a unique numeric book identifier constitutes the group, publication, title, parity check bit and music. Digital information is stored in the form of binary digits and these digits are segmented into blocks. Each of these blocks are appended with parity bit, $a[n+1]$ at the end and it is calculated using the modular operation given below,

$$a\,[n+1] = (a[0] + a[1] + \cdots . a[n])\ mod\ 2 \quad (1)$$

where a is the block of data and n is the number of bits. If the records of the people are stored using Social Security Number (SSN) as the key, the array needed is in the size of 10^{10}. The easiest and the most efficient method would be to store the records in the table using the index a_k which is defined by,

$$a(k) = k\ (mod\ N) \quad (2)$$

where N is the size of the array. The Gauss' Easter formula is also derived using modular operation. The random numbers a_k of a sequence can be generated using convergence theorem,

$$a_{k+1} = \big((x\ \times\ a_k) + y\ \big) mod\ N \quad (3)$$

where a_0 is the seed and the values of x and y are chosen appropriately such that, $0 < x < N$, $0 \leq y < N, N > 0\ and\ GCD(N, y) = 1$.

II. Literature Survey

The paper is written taking into account many technical papers and text books. An attempt is made to implement modular algorithms, both reduction and multiplication. The timing and the extent to which the device is utilized is compared which is also the measure of area needed by the algorithm. The efficient modular multiplier uses two multiple-precision multipliers. With the absence of pre-computation multiplication, the hardware is reduced to one single precision multiplier and an adder. Based on Barrett reduction and Montgomery modular reduction, two interleaved modular multiplication algorithms are implemented. The proposed algorithm outshines the standard modular multiplication by more than 50% in case of speed [1]. Extended Binary GCD algorithm and the modular multiplication algorithm is modified and merged to propose a new modular multiplication or division algorithm. Among the various algorithms currently being used, these two happen to be most compatible as the resources like registers and the combinational logic involved in the operation and controlling the operation can be shared. With Verilog as the description language, the design is implemented in the Synopsys design Compiler using 0.35μm CMOS 3-metal technology [2]. The designed is to be utilized in a multi-core system. The aim was to scale down the data transfers between the cores. Hence a significant improvement was observed when the modular operations where implemented in a 4 core system when compared to a 2 core system [3]. The proposed design is an integration of three algorithms, Kasturba multiplication algorithm, Barrett reduction and Montgomery modular multiplication. It was noted that the designed algorithm required less speed. Parallel implementation of the algorithm appeared to be more advantages. The study speculates that the proposed design gives greater advantage while implementing in a hardware and software platform. The Kasturba multiplication algorithm can be replaced by Toom Cook algorithm [4].

III. Algorithm Description

In the paper, three modular reduction and one modular multiplication algorithms are implemented. The algorithms implemented are:

- Barrett Reduction,
- Mod without Division,
- Montgomery Reduction,
- Montgomery Multiplication.

The Barrett reduction algorithm is based on Equation 4.

$$y = \left\lfloor \left(\frac{j}{b^{k-1}}\right)\left(\frac{b^{2k}}{u}\right)\left(\frac{1}{b^{k+1}}\right) \right\rfloor \quad (4)$$

The main object of the algorithm is to find the value of y that is given by j mod u. The value of β is pre-computed and is given by the Equation 5.

$$\beta = \left\lfloor \frac{b^{2k}}{u} \right\rfloor \quad (5)$$

This is beneficial in instances when several reductions are to be calculated using a single modulo. In this algorithm, the value of base, b is very important, it signifies the base of the inputs. The condition while selecting the base is that it should be greater than 3. This makes the entire computation less complex. Table 1 is the pseudo code for Barrett reduction. The algorithm includes basic operations like addition and multiplication.

Mod without division is a modular reduction algorithm which efficient for computing large modulo operations. The data is streamed and had good spatial and temporal locality. The condition for mod without division is the size of the input should greater than that of the modulo. The algorithm also has a pre-computed β. The algorithm has certain predefined operations as explained below.

- ϒ splits the input j into groups of width u,
- N is the number of groups numbered from 0 to N-1,
- β is the correction factor calculated by $\beta = 2^{width(u)} \bmod u$ (6)

Since in any modular arithmetic operations the resultant can never be greater than modulo value, a while loop is incorporated at the end of the algorithm. The while loop is executed until the value of r is less than u.

The $N-1^{th}$ group is left shifted width of u times. At each shift, if the MSB bit of ϒ is high, the correction is done by setting the MSB bit to zero and adding the pre-computed value β to it. MSB bit is checked till the MSB bit is reset. This entire operation is repeated till N becomes 1. When there is an overflow, it means that ϒ value is greater than u. By adding β to ϒ, it can be ensured that value of ϒ is less than u. The 0th group is added to the final result and again checked for overflow condition. The key point of the algorithm is that the input data is read only once thus eliminating excess processor cycles. The while at the end of the algorithm is to ensure that the final value is not greater than u which is not possible in modular operation. Table 2 is the pseudo code of Mod without Division.

Table 1 Pseudo code for Barrett Reduction.

Inputs: $j = (j_{k-1}j_{k-2} \cdots j_2 j_1 j_0)$ $u = (u_{k-1}u_{k-2} \cdots u_2 u_1 u_0)$ such that $u_{n-1} \neq 0$
Outputs: y = j mod u
Steps : 1. $\beta = \left\lfloor \frac{b^{2k}}{u} \right\rfloor$ 2. $q_1 = \left\lfloor \frac{j}{b^{k+1}} \right\rfloor$ 3. $q_2 = \left\lfloor \frac{q_1 \times \beta}{b^{k+1}} \right\rfloor$ 4. $r_1 = j \bmod b^{k+1}$ 5. $r_2 = (q_2 \times u) \bmod b^{k+1}$ 6. $r = r_1 - r_2$ 7. if $r < 0, r = r + b^{k+1}$ 8. while $r \geq u, r = r - u$

Montgomery reduction is implemented as a part of Montgomery modular algorithm to compute the value of y is given by $tR^{-1} \ mod \ u$. R is given by $R = b^k$ (7) where b is the base of the inputs and k is the width of the inputs in bits. The value of $u' = -u^{-1} \ mod \ b$ (8) is a pre computed value which can be computed by using any of the inverse algorithms like Extended Euclidean algorithm. The algorithm includes simple arithmetic and logical operations like shift, addition and multiplication. The entire algorithm is repeated k times which is equal to the length of the input in bits. Since the final value can never be greater than u, a while loop is employed at the end of the algorithm. Table 3 is the pseudo code for Montgomery reduction.

Table 2 Pseudo code for Mod without Division.

Inputs: $j = (j_{2k-1}j_{2k-2} \cdots j_2 j_1 j_0)$ $u = (u_{k-1}u_{k-2} \cdots u_2 u_1 u_0)$

Steps:
ϒ = Split (j, Width(u))
N = Num(ϒ) – 1
$\beta = 2^{width(u)} \bmod u$
while N > 0
H = ϒ [N]
for (i = 0 to Width(u)-1)
H = H << 1
while $H_{width(u)-1} == 1$
$H_{width(u)-1} = 0$
H = H + β
ϒ[N-1] = ϒ[N-1] + H
if ($ϒ[N-1]_{width(u)-1} == 1$)
$ϒ[N-1]_{width(u)-1} = 0$
ϒ[N-1] = ϒ[N-1] + β
N = N – 1
while ϒ[0] > u
ϒ[0] = ϒ[0] – u

j mod u = ϒ[0]

Table 3 Pseudo code for Montgomery Reduction.

Inputs: $t = (t_{k-1}t_{k-2} \dots t_2t_1t_0)$ $p = (p_{k-1}p_{k-2} \dots p_2p_1p_0)$
Steps: $p' = -p^{-1} \bmod b$ $a = t\ (a = a_{k-1}a_{k-2}a_2a_1a_0)$ for $i = 0$ to $k-1$ $u_i = a_i p' \bmod b$ $a = a + u_i p b^i$ $a = a/b^k$ while $a \geq p$, $a = a - p$
$tR^{-1} \bmod p = a$

Montgomery multiplication is an integral part of Montgomery modular operation. The algorithm is executed prior to the reduction process. The resultant of the algorithm, y is given by $y = edR^{-1} \bmod u$ (9). R is related to the base of the inputs and is defined as per Equation (7). The algorithms also computed the u' using any of the inverse algorithms like Extended Euclidean algorithm. The algorithm consists of basic operations like multiplication, shift and addition. In modulo operation, the resultant cannot be greater than zero, hence a while loop is used at the end of the algorithm to ensure that the value is always less than u. The algorithm is implemented k number of times which is equal to the width of the inputs in bits. Equation 10 includes the operation mod b, since b is the base of the inputs, this operation is equivalent to preserving the last digit of the result. Similarly the Equation 11 includes the divide by b operation, this is equivalent to preserving all the digits except the last digit of the resultant. This reduces the complexity of the algorithm. Table 4 is the pseudo code for Montgomery Multiplication.

Table 4 Pseudo code for Montgomery Multiplication.

Inputs: $e = (e_{k-1}e_{k-2} \dots e_2e_1e_0)$ $d = (d_{k-1}d_{k-2} \dots d_2d_1d_0)$ $u = (u_{k-1}u_{k-2} \dots u_2u_1u_0)$
Steps: $u' = -u^{-1} \bmod b$ $t = 0\ (t = t_{n-1}t_{n-2}t_2t_1t_0)$ for $i = 0$ to $k-1$ $a = (t_0 + (e_i * d_0)\, u') \bmod b$ (10) $t = (t + e_i d + a_i u)/b$ (11) while $t \geq u$, $t = t - u$
$edR^{-1} \bmod u = t$

IV. Results and Discussion

The algorithms are implemented in Xilinx ISE® Design Suite 14.2 using Spartan 6 as the family. The design is synthesized using the XST tool and from the synthesis report and the device utilization summary, the timing and the area utilized by the design is tabulated. Table 5 is the synthesis report of all the implemented algorithms.

Table 5 Timing and area report of the algorithms.

	Timing				Area		
	Minimum input arrival time before clock (ns)	Maximum output required time after clock(ns)	Minimum Period (ns)	Maximum combinational path delay (ns)	Slice Registers	Slice LUT's	IOB's
Barrett Reduction	189.102	4.296	3.365	No Path	18	2770	64
Mod without Division	121.680	4.174	4.030	No Path	92	1558	65
Montgomery Reduction	33.718	5.782	2.117	39.014	2	489	68
Montgomery Multiplication	45.310	5.965	3.384	50.539	2	122	68

The area is given in terms of number of slices, LUT's (Look up Table) and IOB's (Input/ Output Block) utilized. Xilinx reports different types of delays as listed below,

- Minimum period,
- Maximum input arrival time before clock,
- Maximum output required time after clock,
- Maximum combinational path delay.

The device utilization summary that is generated in Xilinx, gives an account of the resources of the FPGA used which are,

- Slices Registers,
- Slices LUT's,
- Bonded IOB's.

V. Conclusion

The modular reduction and multiplication algorithms have merits and demerits of their own. The advantage of Barrett reduction and mod without division is that it involves simple arithmetic operations like addition, multiplication and shift operation. The disadvantage of mod without division is the fact that it cannot be implemented using inputs which are of equal width. The width of j should always be greater than u. The demerit of Barrett reduction is that the base of the inputs should always be greater than 3. Hence it can successfully implemented using binary inputs as their base is 2. In the paper, the base was considered to be 16. The usage of Montgomery Multiplication and Montgomery Reduction can be done in the Montgomery modular operations. This eliminates three 16x16 multipliers and two 31/16 dividers, which can help to reduce the timing and area of the design.

The designs can be incorporated in RSA encryption algorithms and other encryption algorithm. Since the design was implemented in Spartan 6 which is built on proven 45nm technology, the designs can further be designed using 32nm technology. The pre computational values reduce the flexibility of the algorithms. By eliminating these values, the algorithms can definitely made flexible and efficient. The Extended Euclidean algorithm can be replaced with a more efficient algorithm.

References

[1]. Xiaodong Yan and Shuguo Li, "Modified Modular Inversion Algorithm for VLSI Implementation", 2007 7th International Conference on ASIC, 22-25 Oct. 2007.
[2]. Mark A. Will and Ryan K. L. Ko, "Computing Mod Without Mod", Cryptology ePrint Archive, Report 2014/755, 28 Sep 2014.
[3]. Scott Vanstone, Alfred Menezes and Paul van Oorschot, Handbook of Applied Cryptography, CRC Press, Inc. Boca Raton, FL, USA, 1996.
[4]. Satyanarayana Vollala, B. Shameedha Begum and N. Ramasubramanian, Hardware Design for Multiplicative Modular Inverse, 2015 International Conference on Computing and Network Communications (CoCoNet'15), 16-19 Dec., 2015.
[5]. Ankush Yete, Ananya Kajava P, Hazel Melita Rodrigues, Namratha P and Kiran Kumar V.G, Implementation of Montgomery Modular Multiplication using High Speed Multiplier, 2013 International Journal of Current Engineering and Scientific Research (IJCESR), 7-8 Dec. 2013.

Mishal Jasmine Ferrao" Implementation of Modular Reduction and Modular Multiplication Algorithms" IOSR Journal of VLSI and Signal Processing (IOSR-JVSP) , vol. 8, no. 6, 2018, pp. 34-38.