Oregon State University

Electrical & Computer Engineering Department

ECE 575

Project Report

A Modulo Multiplication Hardware Design

By

Adnan Gutub

For

Professor Koc

Winter 2000

Abstract

Several modular multiplication algorithms have been reviewed. One modified modulo multiplication algorithm is chosen to be designed in simple hardware components. The proposed design is shown in blocks of the basic modules and the connections required between the basic components are shown in some details.

Introduction

The RSA cryptographic scheme is a popular secure method used for public key cryptosystems. RSA algorithm is mainly concerned about the computation of modular exponentiation operations. The modular exponentiation is calculated by repeating modular multiplication operation a number of times (as the number of bits of the key). Each computation of modular multiplication consumes a large amount of time, which made the RSA algorithm un-practical for normal encryption and decryption processes [5]. The modular multiplication process can either be implemented in software or hardware. Software implementations, however, are very slow compared with hardware ones.

```
                    Inputs: $X$, $Y$ & $n$  ($X,Y < n$)
                    Result:   $P = X \times Y \bmod n$
                    $k$: is the number of bits in $X$ ;
                    $i = k - 1$ ;
                    $X_i$ :  is the $i^{th}$ bit of X.
1.  $P := 0$
2.  $P := 2P$ ;
3.  If   $P \geq n$ THEN:  $P := P - n$ ;
4.  If   $X_i = 1$ THEN:
5.  {  $P := P + Y$ ;
6.  If   $P \geq n$ THEN:  $P := P - n$ ;  }
7.  $i = i - 1$ ;
8.  If  $i < 0$  THEN GOTO 2 ;
9.  RETURN P ;
```

Figure 1: The modified modulo multiplication algorithm

Modular multiplication can be hardware implemented in different ways. Residue number systems, for example, have been recently receiving more attention [2]. Reported implementations, however, have ignored important hardware implementation issues practically the need for conversion between binary and residue numbers. This has discouraged researchers from using such an approach [3]. Another approach is based on look-up tables, i.e. ROM-based methods. This method, however, is quite expensive since the required memory space grows exponentially with the word size. [3]

The straight-forward method to compute modulo multiplication is by performing the multiplication and then subtracting the modulus several times until the result is less than the modulus. This approach is inefficient and suffers from low speed. This can, however, be improved by merging the modulo subtraction with the multiplication add operations as shown in Figure 1 [1].

```
Initialization steps:
1.  Choose $R > n$, such that $R = 2^k$;  k is the number of bits in n; accordingly R is relatively prime to n
2.  Compute $R^{-1}$ and n' , such that $R.R^{-1} \bmod n = 1$; and  $n' = -n^{-1} \bmod R$
3.  Transform x & y to x' & y' (Montgomery's representation); where: $x' = xR \bmod n$ & $y' = yR \bmod n$

Then compute $z' = MP(x', y') = x'y'R^{-1} \bmod n$
Montgomery modular multiplication steps: MP(x',y')
1.  P    $= x'.y'$
2.  U    $= P + n * (P * n' \bmod R)$
3.  S    $= U/R$
4.  if   $S < n$     then     $MP = S$
5.  else                      $MP = S - n$

Final step:
1.  Transform z' to the normal representation z; where $z = z'R^{-1} \bmod n = MP(z', 1)$
```

Figure 2: Montgomery's modular multiplication method to compute: x.y mod n.

In 1985, Peter Montgomery proposed a new method for modular reduction without regular division. His method has been widely adopted for use in RSA implementations. Montgomery's technique [4], as shown in Figure 2, transforms the normal numbers into another region. In that region, the modulo computations are computed. Then, the results are re-transformed into normal region. Therefore, Montgomery's method requires some initialization steps to perform the modulo multiplication without trail division. This made the decision of this report to choose the modified modulo multiplication algorithm (shown in Figure 1) to be designed in hardware.

Designing Assumptions

The algorithm in Figure 1 is re-outlined, for simplicity, as a graphical flowchart shown in Figure 3. This algorithm assumes that all numbers are in binary bits and the size of the numbers n, x, y, is assumed to be b-bits. The first clock cycle will be assumed as a loading cycle. All the bits of x, y, and n, are loaded in this cycle. The loading is done in a parallel manner, this parallel loading will be clarified later, in the registers & controller section.

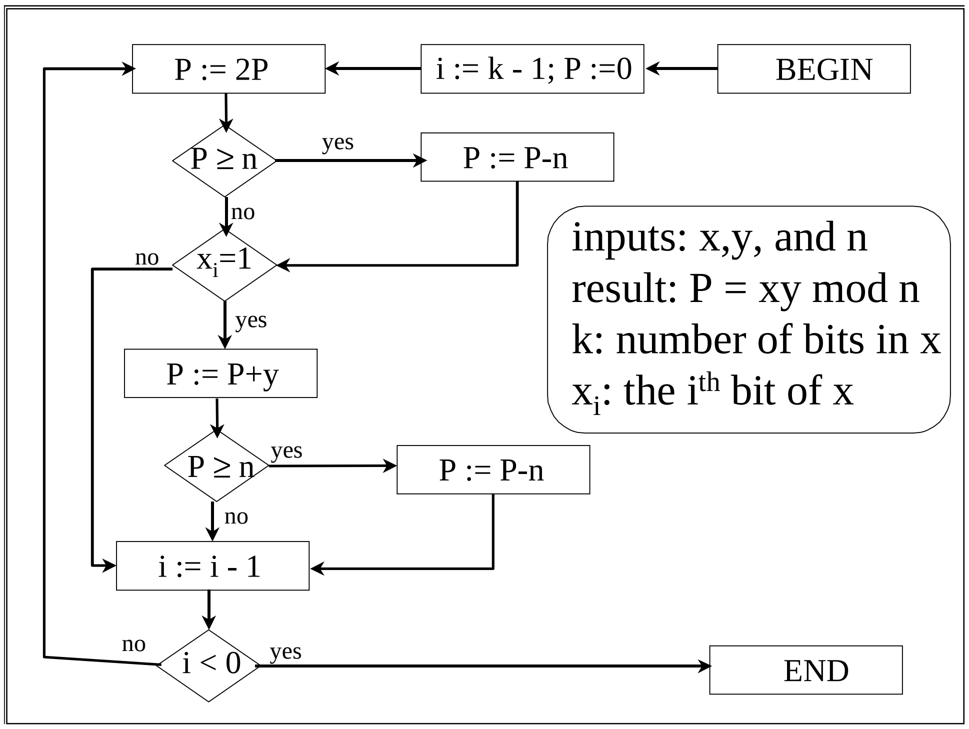

Figure 3: The modified modulo multiplication flowchart algorithm.

The algorithm in Figure 3 shows three large arithmetic operations, two subtractions and one addition. These add/subtract operations can be performed serially using one adder. However, in order to gain the maximum speed, the design assumed to use three different parallel modules. The modules are an adder and two subtractors.

The comparison between P & n, if: P ≥ n is needed twice. This comparison result is accomplished by the observing the subtraction of P-n output-carry-bit. If the output-carry-bit is one, then P ≥ n is true. However, If the output-carry-bit is found to be zero then P is less than n.

The complete modulo multiplication hardware is divided into the Registers & Controller, and the modulo reduction unit. Figure 4 shows a block diagram of the complete modulo multiplication hardware design.

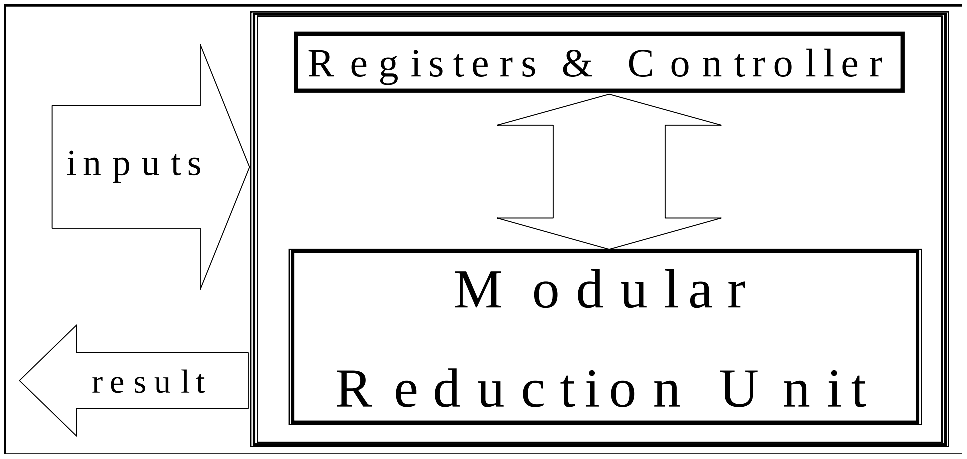

Figure 4: the modular multiplication hardware outline

The Registers & Controller

The controller is mainly the counter ‘i’. The registers are to hold the data of x, y, n, and P. All registers load the data bits in a parallel manner, i.e. all bits are loaded at one clock cycle, however, they differ in the way they release these data bits. One register type is only for ‘x’. It is a parallel load shift register, which feeds the modular reduction unit with one-bit of ‘x’ at a time every clock cycle. The other register type is for data: y, n, and P. This type is pumps all the bits in parallel at each clock cycle.

The Modular Reduction Unit

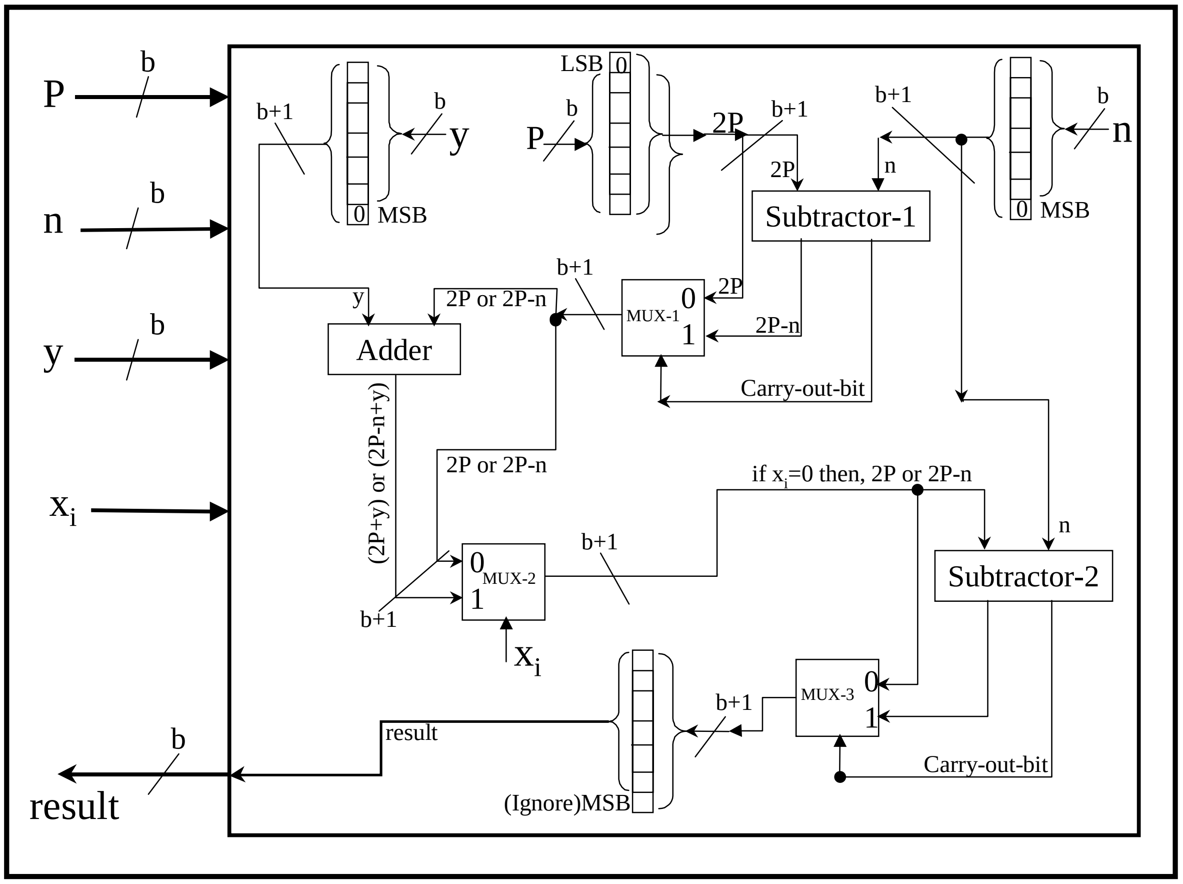

Figure 5: the modular reduction unit

Figure 5 shows the modular reduction unit, which is the basic hardware designed to perform the algorithm shown in Figure 3. The inputs to this hardware unit are P, n, y, and x_i, as found in the algorithm. The bits of 'P' are made zeros in the registers & controller unit. This is performed at the first clock cycle, while loading the bits of x, y, and n. After the loading cycle, the registers and controller unit will feed the modular reduction unit with the data bits of P, n, y, and x_i. Note that the bits used for P and x_i are new for each iteration, while the rest of data are the same. Each iteration requires one clock to be performed.

To clarify the processing of the hardware, consider the algorithm shown in Figure 3. First, 'P' is to be multiplied by 2. This is achieved in hardware by left-shifting 'P' for one-bit, adding a zero bit as the least significant bit (LSB). The shifting process causes

the result of ‘2P’ to be (b+1)-bits, which requires (b+1)-bits subtractor to perform the (P-n) operation. An extra zero-bit is appended to ‘n’ as the MSB (most significant bit).
The algorithm has three comparisons, which are modeled in hardware using the multiplexers: Mux-1, Mux-2, and Mux-3. Mux-1 and Mux-3 are to perform the decision of the comparison: $P \geq n$. The Mux-control-bit is the carry-out-bit of the subtractor. Note that carry-out-bit is one only when ‘P’ is greater than ‘n’, which will make the Mux to choose the P-n value. Mux-2 depends on the x_i bit. If x_i is one, Mux-2 will select the output of the adder to be processed. If x_i is zero, Mux-2 will pick the previous value and the output of the adder will be neglected.
The result is only the b-bits coming out of Mux-3, ignoring the MSB, as shown in Figure 5. Then the result will be stored as ‘P’ in the registers & controller unit to be adopted in the next clock cycle.

Conclusion

Various modular multiplication methods have been surveyed. One modified technique was preferred and modeled in an understandable hardware design. The proposed design is shown as brief blocks. The important connections required between the basic components are shown in some details.

References

[1] G. Orton, M. Roy, P. Scott, L. Peppard & S. Tavares, "VLSI Implementation of Public-Key Encryption Algorithms," *in Advances in Cryptology: CRYPTO'86 Proceedings,* A.M. Odlyzko, Ed. 1987, pp. 277-301.

[2] E. F. Brickell, "A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography," *Advances in Cryptology: Proceeding of CRYPTO'82*, Plenum Press, 1983, pp 51-60.

[3] C. Wu and Y. Chou, "General Modular Multiplication by Block Multiplication and Table Lookup," *IEEE international Symposium on Circuits and Systems, ISCAS'94*, London, UK: IEEE, 1994, pp 295-298.

[4] Cetin Kaya Koc, Tolga Acar, and Burton S. Kaliski, Jr. "Analyzing and Comparing Montgomery Multiplication Algorithmsm," *IEEE Micro*, June 1996, pp 26-33.

[5] William Stallings, "Cryptography and Network Security: Principles and Practice," *Prentice-Hall, Inc*. Upper Saddle River, New Jersey, 2nd edition, 1995.