DESIGN METHODOLOGY FOR BOOTH-ENCODED MONTGOMERY MODULE DESIGN FOR RSA CRYPTOSYSTEM

Jye-Jong Leu and An-Yeu Wu
Department of Electrical Engineering,
National Central University, Chung-Li, 320, Taiwan, R.O.C.

ABSTRACT
In this paper, a design methodology for the design of Montgomery module is proposed. We summarize the result in pseudo C-like codes and call it Booth-encoded Montgomery modular multiplication algorithm. Using this algorithm, iteration number is reduced to about n/2 in each Montgomery operation. In addition, we apply the folding and unfolding techniques to shorten the critical path. Finally, we propose the 4-bit-digit-serial pipelined architecture to process RSA En/Decryption in a more efficient way. The speed of the proposed algorithm is approximately 1.7 times of the most RSA VLSI designs based on original Montgomery modular multiplication algorithm.

1. INTRODUCTION
As the use of network continues to grow, it has more impact on our daily life. Now many jobs, such as shopping, remittance, telephony, etc., can be done in front of a computer through network. To protect important data from the invasion of network hackers, the security of network becomes an essential issue. Recently, the public key cryptography becomes very popular due to its flexibility. The most well-known public key cryptography is RSA cryptosystem, which is named after its inventors, Rivest, Shamir and Adleman [1]. In the realization of RSA cryptosystem, long word-length (= 512 bits) is usually employed to meet the security requirement. Hence, it calls for very large silicon area in VLSI implementations. The speed performance is limited by its long wordlength, too. Fast exponential computation now becomes increasingly important for its wide use in RSA encryption. Many methods, such as H-algorithm, L-algorithm, etc., are proposed to accelerate the exponential computation. Besides, most recent RSA designs employ the Montgomery modular multiplication algorithm [3][8] as kernel operation in high-performance exponent-computation algorithms. It also plays an important role to improve the efficiency of RSA En/Decryption operation. Nevertheless, the speed of a 512-bit RSA En/Decryption chip is still far slower than network transmission bandwidth today.

In this paper, we propose a design methodology for Booth-encoded Montgomery modular multiplication algorithm. It can reduce the iteration number to about n/2 in each Montgomery operation. It uses similar hardware complexity compared with most novel design [3][6]. Our goal is to maximize the speed of the encryption conversion with acceptable hardware complexity in a systematic approach.

2. MODULAR EXPONENTIAL ALGORITHMS
The RSA public-key cryptography performs the computation of

\[ C = M^e \pmod{N}. \] (1)

In many RSA implementations, for the exponent \( E \) with long word-length, \( H \) algorithm and \( L \) algorithm are widely used to reduce the computational complexity. These two algorithms perform modular exponentiations by using squaring and multiplying cyclically. The detailed operations of the \( L \) and \( H \) algorithms can be found in the previous works [4]. In both \( H \) and \( L \) algorithms, the computation of

\[ R = A \cdot B \pmod{N} \] (2)

plays a very important role. Montgomery's modular multiplication algorithm [5] is usually employed to perform the operation of Eq. (2). Note that Montgomery's algorithm cannot be directly applied to modular exponential multiplication algorithm. This is because Montgomery's algorithm performs the computation of

\[ A \cdot B \cdot 2^k \pmod{N} \cdot (kN). \] (3)

The extra factors of \( 2^k \) and \( k \) are inconsistent with Eq. (2). To eliminate the extra factors \( 2^k \), a pre-computing process is required. Then, to solve the residue problem of factor \( k \), a post-computing process is needed. For a single modular multiplication, it required three Montgomery modular multiplication processes to complete the whole operation. A modified modular exponential algorithm using \( H \) algorithm is summarized in Table 2, in which only one pre-computing process and one post-computing process are needed. Thus, the Montgomery module dominates the overall speed performance of RSA chip as well as the hardware complexity in VLSI design.

3. DESIGN METHODOLOGY
To improve the speed performance, we propose a novel design methodology for Montgomery module. Figure 1(a) shows the architectural design based on original Montgomery module multiplier algorithm. The iterations required by this architecture are \( n \). The procedure is as follows:

- **Step (a):** Take loop-unrolling technique on this architecture, and now the number of iteration is reduced to \( n/2 \). The result is shown in Fig. 1(b).
- **Step (b):** Change the sequence of addition. See Fig. 1(c).
- **Step (c):** Take radix 4 Booth-encoding technique on first two rows of adders, and encode the input data \( \{a_{i-1}, a_i, a_{i+1} \} \) to Booth code [10]. The result is shown in Fig. 1(d).
- **Step (d):** Combine the last two rows of adders, and create a lookup table. See Fig. 1(e).
- **Step (e):** Simplify the table. The final architecture is shown in Fig. 1(f).

The design methodology helps to derive low-latency and high-
speed architecture for the design of Montgomery module. Finally, the operation are summarized in pseudo C-like code (Table 1), and we call it Booth-encoded Montgomery modular multiplication algorithm. The implementation of the Modulo selector (sel()) function is very simple. Only two logic gates are needed to construct it, and it is illustrated in Fig. 2. After the operation of sel() function, the output of the 2 least significant bit (LSB) of $P[i+1]$ must be zero valued. Then, we can remove these 2 LSBs by simply shift $P[i+1]$ 2 bit right, i.e. divide $P[i+1]$ by 4. With this algorithm derived by the design methodology, We need only $n/2$ iterations while the hardware complexity are about the same as existing approaches [3] [6].

```
Booth(A)
{
    a_i=0; a_o=0; a_n=0;
    for (i=0; i<=n; i=i+2)
    {
        switch(a_i, a_o, a_n)
        {
            case 0: c_{i-1}=0; break;
            case 1: c_{i-1}=1; break;
            case 2: c_{i-1}=1; break;
            case 3: c_{i-1}=2; break;
            case 4: c_{i-1}=2; break;
            case 5: c_{i-1}=1; break;
            case 6: c_{i-1}=1; break;
            case 7: c_{i-1}=0; break;
        }
    }
    return C=[c_{n-2} c_{n-1} c_{n-2} c_{n-3} ... c_1 c_0]
}

Sel(q_n)
{
    q_i=q[0]; /* q0=[q_1, q_0] */
    if (n_i=0)
        r_i = q[1] @ q[0];
    else
        r_i = q[1];
    return (r_i, r_0);
}

M3(A,B,N)
{
    Cn = Booth(A)=[c_{n-2} c_{n-1} c_{n-2} c_{n-3} ... c_1 c_0];
    P[0]=0;
    for (i=0; i<=n; i=i+2)
    {
        q_i=(P[i-1]+c_i * B) mod 4;
        P[i]=P[i-1] + c_i * B + Sel(q_i, n_i) * N div 4;
    }
    return P[n/2+1];
}
```

Table 1: Booth-aid Montgomery's algorithm.

4. DESIGN OPTIMIZATION

We design the Montgomery module based on the proposed Booth-encoded Montgomery's algorithm. The design of detailed circuit is shown in Fig. 2. To illustrate our design, we use an 8-bit Montgomery unit in Fig. 3 as an example. The Montgomery module uses the modular $N$, modular $3N$, multiplicand and Booth-encoded multiplicator as the input data, and it can complete the Montgomery modular multiplication in one single clock. Note that by using the Booth-encoding technique, the partial output in each stage, which contains two rows of full adders, may be negative. Thus, some sign-extend circuits are required. By applying these operations to all stages, one Montgomery modular multiplier can be formed. Then, we can obtain the output data, i.e., $R[9:0]$, from the bottom of the structure. To accelerate the speed of operation, the pipelining stages can be inserted to shorten the critical path. We add pipeline stages to the unfolded architecture of Fig. 3, where the dotted lines represent the pipe-

---

**Fig. 1: Design methodology of Montgomery module**

**Table 2: Pseudo C program of modified modular exponentiation algorithm, where $M$: message, $E$: exponent, $N$: modulus, $C$: constant, and $R$: result.**

\[
M' = M3(M, C, N);
R[0] = M';
\]

For($i = 0; i < k-1; i++$)
{
    $R[i+1] = M3(R[i], R[i], N)$;
    if($R[k-1]-2 = 0$)
        $R[i+1] = M3(R[i+1], M', N)$;
    $R[k-1] = M3(N, R[k-1], N)$;
    Return $R[k-1]$;
}

\[
\text{Table 2: Pseudo C program of modified modular exponentiation algorithm, where } M: \text{ message, } E: \text{ exponent, } N: \text{ modulus, } C: \text{ constant, and } R: \text{ result.}
\]

V-358
lined stages. The 8-bit pipelined architecture of Montgomery module is illustrated in Fig. 4. Figure 5 shows the detailed circuit of each M-cell in Fig. 4.

![Architecture Diagram](image)

**Fig. 2:** The architecture of the Montgomery module.

### 4.1 Digit serial architecture

By extending the 8-bit design, we can have the complete 512-bit Montgomery module as illustrated in Fig. 6. The input data, including multiplier, multiplicand, R, N, 3N, and control signal P, are all fed 4-bit digit serially in each clock from the left side. The outputs can be obtained in 4-bit digit serial way in each clock from the right side, too. Hence, it is a 4-bit digit-serial architecture, which contains 257 M-cells.

![Module Diagram](image)

**Fig. 3:** Overall structure of the unfolded 4-bit Montgomery unit based on Booth-encoded Montgomery's algorithm.

![Diagram](image)

**Fig. 4:** Digit-serial 8-bit Montgomery module after the folding process.

The 512-bit Montgomery unit requires a total of (256+1) M-cells to construct the whole unit. It takes 1*(256+1) clock cycles (1 clock cycle/each cell) to complete the 512-bit Montgomery operation. The total clock cycles of one 512-bit RSA operation that will be performed in this module is

\[(\log_2 E + v(E)) * 1 * (256+1),\]

where v(E) is the number of nonzero bits in the exponent. Hence, it takes about 1.5 * 512 * 1 * 257 = 0.19M clock cycles for the average cases, or 2 * 512 * 1 * 257 = 0.27M clock cycles for the worst case in a single 512-bit RSA En/Decryption operation.

![Detailed Circuit](image)

**Fig. 5:** Detailed circuit of the proposed Montgomery cell.

![Diagram](image)

**Fig. 6:** Digit-serial 512 bits Montgomery unit.

### 4.2 Improved module utilization

By examining the Montgomery unit in Fig. 6, we find a potential disadvantage that only half of total 257 M-cells are running at the same time. It means that almost 1/2 of total cells are idle as illustrated in Fig. 7. Hence, to increase the efficiency, we add a new data loop and a multiplexer circuit to reuse those M-cells (see Fig. 8). By doing so, we can reduce the cell number by 2. In other words, only 129 M-cells are required now. As a result, we can significantly reduce the required hardware complexity. The gate count of our design is listed in Table 3. The design by Yang et al. [3] is also shown for comparison purpose. We can see from the Table 3 that the hardware complexities of these approaches are about the same. However, the speed performance of our design is approximately 1.7 times of the designs in [3][6] using H algorithm.

### 4.3 The RSA processor design

Fig. 9 shows the overall block diagram of our 512-bit RSA processor. We use the modified modular exponential algorithm and Booth-encoded Montgomery’s algorithm to implement our design. The advantages of our design are as follows:

- **Scalable design:** The bit number of a RSA processor corresponds to the number of M-cells. We can construct RSA processor with different bit number by changing the number of M-cells.
- **Comparable Speed Performance:** In our design, we can shorten the critical path to 8.5ns. If we add 1ns delay as the design margin for more conservative estimation, the architecture can deliver a baud rate of 200kbit/sec for the worst case. Compared with other designs using H-algorithm, we find that this architecture
has the best performance. Table 3 and Table 4 show that our design has comparable speed performance but with lower hardware complexity.

![Fig. 7: Timing diagram of data-flow in Montgomery module.](image)

![Fig. 8: Overall scalable digit-serial Montgomery module.](image)

![Fig. 9: Architecture of the proposed 512-bit RSA processor.](image)

<table>
<thead>
<tr>
<th>Design</th>
<th>DFF</th>
<th>FA</th>
<th>AND</th>
<th>MUX</th>
<th>XOR</th>
<th>Total gate Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yang et al.[3]</td>
<td>12n</td>
<td>2n</td>
<td>3n</td>
<td>8n</td>
<td>N/A</td>
<td>74,493</td>
</tr>
<tr>
<td>Our Design</td>
<td>12.3n</td>
<td>2n</td>
<td>4n</td>
<td>3n</td>
<td>1.3n</td>
<td>77,892</td>
</tr>
</tbody>
</table>

Table 3: Comparison of gate counts.

5. CONCLUSIONS

A design methodology for high-speed Montgomery module design is proposed and a novel Montgomery modular multiplication algorithm is derived. A new 4-bit digit-serial RSA chip design based on this algorithm is presented. By comparing with other designs, we can see that the Booth-encoded Montgomery modular multiplication algorithm provides a better choice for RSA chip design. The digit-serial and scalable features of our design make it a very good candidate for large word-length RSA chip implementation.

<table>
<thead>
<tr>
<th>Designs</th>
<th>Year</th>
<th>Gate count</th>
<th>Bit per chip</th>
<th>#of clocks</th>
<th>Technology</th>
<th>Clock Rate</th>
<th>Baud Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Victor[6]</td>
<td>1994</td>
<td>75k</td>
<td>512</td>
<td>0.125M</td>
<td>1um</td>
<td>25M</td>
<td>100k</td>
</tr>
<tr>
<td>NTT [7]</td>
<td>1994</td>
<td>105k</td>
<td>1024</td>
<td>1M</td>
<td>0.5um</td>
<td>40M</td>
<td>20k</td>
</tr>
<tr>
<td>Chen [8]</td>
<td>1995</td>
<td>77k</td>
<td>512</td>
<td>1.05M</td>
<td>0.8um</td>
<td>50M</td>
<td>24.3k</td>
</tr>
<tr>
<td>Yang et al. [3]</td>
<td>1998</td>
<td>74k</td>
<td>512</td>
<td>0.54M</td>
<td>0.6um</td>
<td>125M</td>
<td>118k</td>
</tr>
<tr>
<td>Guo et al. [9]*</td>
<td>1998</td>
<td>132k</td>
<td>512</td>
<td>N/A</td>
<td>0.6um</td>
<td>143M</td>
<td>278k</td>
</tr>
<tr>
<td>Our design</td>
<td>1999</td>
<td>78k</td>
<td>512</td>
<td>0.27M</td>
<td>0.6um</td>
<td>105M</td>
<td>200k</td>
</tr>
</tbody>
</table>

Table 4: Hardware complexity and performance of current RSA designs. *L algorithm

6. REFERENCES


