Efficient Methods in Converting to Modulo $2^n+1$ and $2^n-1$

Kooroush Manochehri
Department of Computer Engineering and IT,
Amirkabir University of Technology, Tehran, IRAN

Saadat Pourmozafari
Department of Computer Engineering and IT,
Amirkabir University of Technology, Tehran, IRAN

Babak Sadeghian
Department of Computer Engineering and IT,
Amirkabir University of Technology, Tehran, IRAN

Abstract

Modulo $2^n+1$ and $2^n-1$ multiplication plays an important role in residue number systems. In this paper two efficient methods for converting the input number to these moduli are presented. One of them has less area than the other but the second one has more processing speed. The diminished-1 representation of numbers are most suitable for multiplication in $2^n+1$ thus for this modulus the input number is converting to that representation.

Keywords: Modular multiplication, Diminished-1, RNS, CSA, Wallace tree

1. Introduction

Residue number system (RNS) has long been considered an alternative to weighted (binary) representation in digital signal processing [1], public key cryptography and especially in RSA implementations [2,3,4]. RNS can be used to represent numbers using independent residues of manageable word length and to exploit the independence of these residues to facilitate parallel computation of public key cryptosystems, as these typically require modular multiplication of very large integers.

In RNS, numbers are represented according to a base $\beta=(m_1,m_2,\ldots,m_k)$ of $k$ relative prime moduli, $k$ is the size of the base. An integer ‘A’ is represented by the sequence $(a_1,a_2,\ldots,a_k)$ of positive integers, where $a_i = A \mod (m_i)$, $i=1\ldots k$. The Chinese remainder theorem (CRT) [5] ensures the uniqueness of this representation within the range $0 \leq A < M$, where $M = \prod_{i=1}^{k} m_i$. A constructive proof of this theorem can be used to convert back from its residue representation:

$$A = \sum_{i=1}^{k} a_i M_i \cdot M_i^{-1} \mod M$$

where $M_i = \frac{M}{m_i}$ and $M_i^{-1} \mod m_i$ is the inverse of $M_i$ modulo $m_i$.

The advantage of RNS is that addition, subtraction and multiplication are simple and can be implemented in constant time on a parallel architecture. If $a$ and $b$ are given in their RNS form $(a_1,a_2,\ldots,a_k)$ and $(b_1,b_2,\ldots,b_k)$ one have

$$A \pm B = [a_1 \pm b_1]_{m_1},[a_2 \pm b_2]_{m_2},\ldots,[a_k \pm b_k]_{m_k}$$

$$A \times B = [a_1 \times b_1]_{m_1},[a_2 \times b_2]_{m_2},\ldots,[a_k \times b_k]_{m_k}$$

The disadvantage of this representation is two folds, one cannot decide whether $(a_1,a_2,\ldots,a_k)$ is greater than $(b_1,b_2,\ldots,b_k)$, and the overflow might happen during calculation is not detectable. From a cryptographic viewpoint, these difficulties are not to be considered as real drawbacks.

The moduli in the form of $2^n+1$ and $2^n-1$ are frequently exploited in RNS [6] because the subtraction, addition and multiplication can be performed simply in these residues. Also calculating residue of a number in these moduli is simple, in $2^n-1$ [7] and in $2^n+1$ if exploiting the diminished-1 representation [8], by a simple addition can obtained the result. Modular multiplication is the one of the operations that uses the RNS for increasing its speed.

If the numbers were in the diminished-1 representation, with Wallace tree [9] and CSA (Carry Save Adder) architecture [10] the modular multiplication in modulo $2^n+1$ can be done in the most efficient method [10]. Also for modular multiplication in modulo $2^n-1$ this architecture can be used without converting to diminished-1. In many of the architectures they focus on increasing the calculating performance and suppose that the input numbers are in appropriate format but always the input numbers are not in this format [11]. For example suppose that the input of a $2^n+1$ modular multiplier come from another multiplier that its output is not compatible with the desired format, so conversion is needed [12].

Two methods for each modulus are presented in this paper. For the first method there is no need to change multiplier modules and only one extra module added in order to convert the input number. In this method the speed is less than the other method due to the full addition, which happens after the CSA addition. In the
second method the multiplier is changed to support the input in CSA format. It has more speed than the first but the area is increased. Note that these methods are presented generally and may be insecure against side channel attack or power analysis.

This paper is organized as follows: In section 2 the diminished-1 representation is briefly reviewed. In section 3 the architecture of multipliers is described. The conversion methods are introduced in section 4 and 5. In section 6 propose ideas for using the studied methods in common architectures and finally in section 7 this paper is concluded.

2. Diminished-1 representation

The diminished-1 representation of numbers was proposed by Leibowitz [8], as a convenient and efficient form for modulo \(2^{n+1}\) operations on binary numbers. Let \(d(A)\) be the diminished-1 representation of \(A\), then:

\[
d(A) = (A-1) \mod(2^{n+1})
\]

The advantage of this representation is that zero is uniquely identified by MSB=1, for which all arithmetic operations are inhibited. Arithmetic operations for this representation are as follows:

\[
d(A + B) = d(A) \oplus d(B) = d(A) + d(B) + 1 \mod(2^{n+1})
\]

\[
d(A - B) = d(A) \oplus [-d(B)] = d(A) + d(B) + 1 \mod(2^{n+1})
\]

\[
d\left(\sum_{k=1}^{n} A_k\right) = \sum_{k=1}^{n} d(A_k) = d(A_1) \oplus d(A_2) \oplus \ldots \oplus d(A_n)
\]

\[
= d(A_1) + d(A_2) + \ldots + d(A_n) + n - 1 \mod(2^{n+1})
\]

In these relations, \(\oplus\) represent addition and \([-x]\) is negation in the diminished-1 representation. \(d(B)\) represents the one’s complement of \(d(B)\) and \(\sum_{k=1}^{n} d(A_k)\) represent modulo \(2^{n+1}\) summation of diminished-1 numbers.

3. Architecture of multipliers

In this section the architectures of modular multiplier for both modulo \(2^{n+1}\) and \(2^{n-1}\) are presented.

3.1. Modulo \(2^{n+1}\) multiplier

Architecture for this purpose is discussed in [10]. This architecture assumes that neither the multiplier nor multiplicand is zero. Zero detection can be done easily by testing the most significant bit (MSB).

The advantage of this architecture is that it uses \(n\)-bits CSA adder instead of \(2n\)-bits. This reduces area and also increases its speed. By ignoring the zero inputs, the result of modular multiplication in diminished-1 format can be obtained by equation:

\[
d(BA) = \left(\sum_{k=1}^{n-1} b_k d(2^k A) \oplus \bar{Z} \oplus d_1(A)\right) + 1 \mod(2^n + 1)
\]

The expression in parenthesis is done by diminished-1 addition. \(Z\) is the number of zeros of the \(n\)-1 bits from \(b_1\) to \(b_k\) (\(b_i\) is \(i\)-th bit of \(B\) from LSB \((b_0)\) to MSB \((b_{n-1})\)).

\(d_1(A)\) can be computed by this formula:

\[
d_1(A) = b_0 d(A) + b_n d(2A)
\]

If \(b_n=1\), \(d(2A)\) is implemented by left rotation together with a complement operation on the input rotated bits.

All operation in this equation are diminished-1 operations except for the final binary addition of 1, which can be easily implemented by setting the carry in of the final adder. For computing \(Z\) a counter can be used as shown in figure 1 for a 7-bit counter.

![Figure 1. The implementation of a 7-bit counter](image1.png)

Architecture for modulo \(2^{n+1}\) multiplier is shown in figure 2. Each of rectangular in this figure represents the FA block. Carry correction is done by complement carry and addition with the first bit.

![Figure 2. Architecture for modulo 2^{n+1} multiplier](image2.png)
3.2. Modulo $2^n-1$ multiplier

The residue of $A$ modulo $2^n-1$ can be computed by

$$ A \mod (2^n-1) = A \mod 2^n + A \div 2^n $$

where div represent division [7]. So for addition in this modulus, the carry must be added to first bit of result (without any complementation). In order to compute $2^n-1$ modular multiplication, this rule can be used and define its formula as follow:

$$ d(BA) = \sum_{k=0}^{n-1} b_k e^k (2^k A) \mod (2^n - 1) $$

If $b_k=1$, $e(2^k A)$ is implemented by left rotation.

This architecture can use Wallace tree and CSA the same as previous multiplier. The difference with previous architecture is:

1) For carry correction there is no need to complement it.
2) There is no need to compute number of zeros.
3) $d_1(A)$ is eliminated.
4) There is no need to detect zero.

Architecture for modulo $2^n-1$ multiplier is shown in figure 3.

![Figure 3. Architecture for modulo $2^n-1$ multiplier](image)

4. First conversion method

A method for converting a number to desired moduli is presented in this section. In this method there is no need to change the multipliers architecture.

4.1. Converting to diminished-1 in $2^n+1$

Since $2^n = -1 \mod (2^n+1)$, the residue operation is accomplished by

$$ A \mod (2^n+1) = A \mod 2^n - A \div 2^n $$

For calculating the remainder of a number in modulo $2^n+1$, the quotient of division with $2^n$ (that may be necessary to calculate its remainder in modulo $2^n$) is subtracted from the remainder of this division, in modulo $2^n+1$. Suppose that $A$ has $k$ n-bit blocks, in order to calculate the remainder of modulo $2^n+1$ these relations are confirmed:

$$ A = (A_{k-1}, A_{k-2}, \ldots, A_1, A_0) $$

$$ R = (A_0 + (A_1 + (A_2 - \ldots - (A_{k-2} - A_{k-1}) - \ldots)) \mod (2^n+1) $$

$$ R0 = A_0 + A_2 + A_4 + \ldots \mod (2^n+1) $$

$$ R1 = A_1 + A_3 + A_5 + \ldots \mod (2^n+1) $$

$$ R = R0 - R1 \mod (2^n+1) $$

$R$ can be acquired from $R0$ and $R1$ but calculation of $d(R)$ or $d(R0-R1)$ is the main goal. Suppose that $x=R0+1$ and $y=R1+1$ and the relation of diminished-1 is taken, then:

$$ d(x - y) = d(x) + d(y) + 1 = R0 + R1 + 1 $$

$$ = d(R0)+1 + d(R1)+1 + 1 = d(R0)+1+2^n-1-d(R1)-1+1 $$

$$ = d(R0) + d(R1) + 1 = d(R0 - R1) $$

So for calculation of $d(R0-R1)$ there is no need to calculate each $d(R0)$ and $d(R1)$. Note that in this relation the fact that $x$ is one’s complement and is equal to $2^n-x$, is used.

Finally for converting a number to diminished-1 in modulo $2^n+1$, first one must calculate $R0$ and $R1$ by doing diminished-1 addition with the appropriate blocks (note that there is no need to converting these blocks to diminished-1 and just suppose that they are in diminished-1 representation and use diminished-1 addition rule for them).

This is true if the number of blocks in $R0$ and $R1$ were equal because the differences between diminished-1 and binary number are eliminated by subtraction. For instance if $R0$ and $R1$ were not diminished-1 and one suppose that they are in diminished-1 representation, each of them is more than desired number. Therefore if each of them has $k$ blocks then it has $k$ values larger than desired and by subtracting ($R0-R1$) these extra values are eliminated.

If they are not equal, a zero block must be added. Therefore the final method can be summarized as follows:

1) Calculate the summation of odd and even blocks with diminished-1 rules and CSA architecture.
2) Add the result of summation of even blocks with the one’s complement of the result of the odd blocks.
3) Perform full addition in final result that is in CSA format.

Figure 4 shows architecture of 6 blocks. This architecture can be used for both inputs. If there is an input less than 6 blocks all other blocks must be zero.
4.2. Converting to modulo $2^n-1$

This method is similar to converting to diminished-1 but some changes must be taken.

Since $2^n = 1 \mod(2^n-1)$, the residue operation is accomplished by

$A \mod (2^n-1) = A \mod 2^n + A \div 2^n$

Suppose that $A$ has $k$ n-bit blocks, in order to calculate the remainder of modulo $2^n-1$ the following relations are true:

$A = (A_{k-1}, A_{k-2}, \ldots, A_1, A_0)$

$R = A_0 + A_1 + \ldots + A_{k-1} \mod(2^n-1)$

It is obvious that there is no need to calculate the summation of odd and even blocks separately. The addition is taken in mod $2^n-1$ and not in diminished-1, so the inverter of previous architecture can be removed. For instance the architecture of first method for 6 blocks is shown in figure 5.

5. Second conversion method

In the first method although the multiplier architecture unchanged but in the final step of conversion, the full addition must take place to convert the CSA format to binary representation so carry propagation uses many times in this step. Now if the multiplier can accept CSA format, without this step the speed definitely increase and due to inputs are doubled, the area increase about 2 times, but due to the property of Wallace tree, the delay of one full adder (or a CSA adder) is added.

5.1. Converting to diminished-1 in $2^n+1$

For modular multiplication the bits of B is needed to calculate the number of zeros so in this method only one of the inputs can be converted and the other one must be in diminished-1. If sequential architecture is used to calculate one addition in each cycle, also the second input can be converted to diminished-1. In sequential architecture for acquiring each bit the BRFA (Barrel Register Full Adder) architecture [13,14] can be used.

For instance of multiplier for the second method if $A$ is a number that is converted and is in CSA format and has these values: $S=0010$ and $C=0110$, also assume that the other input (B) is in diminished-1 representation and has a value equal to 1011, the additions that must take place in Wallace tree are as follows:

$$\begin{align*}
A \times B &= \\
0101 &= b_1d(2A)+d(2S)+d(2C) \\
1101 &= 0\times d(A)+1\times d(2A)+d(2S)+d(2C) \\
0101 &= b_1d(2A)+d(2S)+d(2C) \\
1101 &= b_2d(2^2A)=0 \\
0000 &= b_2d(2^2A)=0 \\
0110 &= b_3d(2^3A)=d(2^2S)+d(2^2C) \\
0100 &= \bar{Z} \text{(Z is number of zeros in B)} \end{align*}$$

It’s obvious that each of the rows replaced with two numbers according to C and S. As mentioned previously for computing $d(2^3A)$ cyclic shift is used and the carry must be complemented. This rule can be used for both C and S simultaneously. The proof is as follow:
\[ d(A) = d(S+C) = d(S) + d(C) + 1 \]

As shown in order to calculate the summation for \( S \) and \( C \) each of them can be shifted separately, so the main architecture may be used except that the number of addition is doubled.

Architecture for converting six blocks to diminished-1 by second method is shown in figure 6. This architecture is like the previous one but two full additions at the final step is removed.

![Figure 6. Architecture of second method for 6 blocks (mod \( 2^n+1 \))](image)

Table 1 shows the result of comparing the first and second method based on area and delay, for converting to diminished-1 and calculate the modular multiplication.

### Table 1. Comparing the two methods for diminished-1 conversion

<table>
<thead>
<tr>
<th></th>
<th>First method</th>
<th>Second method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Delay</strong></td>
<td>[\log_{1.5}K^<em>(\text{CSA delay}) + K^</em>(\text{inverter delay}) + n*(\text{FA delay}) + n*(\text{Half adder delay}) + \text{multiplier delay}]</td>
<td>[\log_{1.5}K^<em>(\text{CSA delay}) + K^</em>(\text{inverter delay}) + \text{multiplier delay} + \text{CSA delay}]</td>
</tr>
<tr>
<td><strong>Area</strong></td>
<td>[\log_{1.5}K^* (n-bits CSA Area) + K^<em>(n-bits inverter Area) + n</em>(FA Area) + n*(Half Adder Area) + \text{multiplier Area}]</td>
<td>[\log_{1.5}K^* (n-bits CSA Area) + K^<em>(n-bits inverter Area) + 2</em>\text{multiplier Area}]</td>
</tr>
</tbody>
</table>

In this table first expressions in bracket are delay or area of conversion block and the other is delay or area of multiplier block. \( K \) represents the number of input blocks. For instance if the input was 6 n-bits blocks the delay of conversion block for second method is 3 CSA delay. As shown the delay of second method is less than the first and its area is about 2 times more than first method. Note that in this comparison the delay and area of inverter gate in multiplier block is ignored and for better comparison assume that just one of the inputs need to be converted.

### 5.2. Converting to modulo \( 2^n-1 \)

This method is similar to the previous method and for each input must assume there are two inputs. Like second method for converting to diminished-1, one of the inputs can be converted and the other one must be in proper format because the bits of \( B \) must be known.

Architecture for converting six blocks to this modulus by second method is shown in figure 7. This architecture is similar to the previous one but two full additions at the final step is removed.

![Figure 7. Architecture of second method for 6 blocks (mod \( 2^n-1 \))](image)

Table 2 shows the result of comparing the first and second method based on area and delay, for converting to modulo \( 2^n-1 \) and calculate the modular multiplication.

### Table 2. Comparing the two methods for modulo \( 2^n-1 \) conversion

<table>
<thead>
<tr>
<th></th>
<th>First method</th>
<th>Second method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Delay</strong></td>
<td>[\log_{1.5}K^<em>(\text{CSA delay}) + n</em>(\text{FA delay}) + n*(\text{Half adder delay}) + \text{multiplier delay}]</td>
<td>[\log_{1.5}K^*(\text{CSA delay}) + \text{multiplier delay} + \text{CSA delay}]</td>
</tr>
<tr>
<td><strong>Area</strong></td>
<td>[\log_{1.5}K^* (n-bits CSA Area) + n*(FA Area) + n*(Half Adder Area) + \text{multiplier Area}]</td>
<td>[\log_{1.5}K^* (n-bits CSA Area) + 2*\text{multiplier Area}]</td>
</tr>
</tbody>
</table>

In this table first expressions in bracket are delay or area of conversion block and the other is delay or area of multiplier block. It’s obvious that this table is similar to table 1 but the inverter is removed. As shown the delay of second method is less than the first and its area is about 2 times more than first method. In this table for better
comparison assume that just one of the inputs need to be converted.

6. Using the proposed methods

For converting to modulo $2^n-1$ one can assume that the input is not in this format and always use conversion step in the input of multiplier, even if the number is in modulo $2^n-1$, recalculation doesn’t change its value. For modulo $2^{n+1}$ using diminished-1 representation, if convert the converted number again, the result will be incorrect and must distinguish the situation that it is necessary to convert the input or not. To handle this problem the following methods can be used:

If the first conversion method is used, for ignoring the conversion and use the unconverted input in multiplier, an addition bit for a multiplexer can be used to choose a direct input or converted input for multiplier.

In the second method due to changes in multiplier, the previous solution cannot be used. By this method must ensure that the input is never in diminished-1 format. For this purpose an addition by zero can be done in the end of multiplication to convert number from diminished-1 to binary number. In diminished-1 zero is representation of number 1 and by this operation the number is incremented by one and the result will not be in diminished-1. By this attendance there are not any diminished-1 number out of the $2^{n+1}$ multiplier block because only the $2^{n+1}$ multipliers produce this number and by proposed method the output of these multipliers will not be in diminished-1 representation.

7. Conclusion

In order to increase the processing speed in RNS the moduli in the format of $2^n+1$ and $2^n-1$ are used. There are many enhanced architectures with improved performance using Wallace tree and CSA. In most of these architectures they assume that the numbers are in an appropriate format. In this paper new methods for converting to modulo $2^n+1$ and $2^n-1$ are presented. With using CSA architecture the computation time is reduced for these conversion blocks.

In the first method due to two n-bits carry propagation adder, conversion needs more time but its output can be used in common multiplier architectures without any changes. Thus the delay and Area of multiplier is unchanged and if conversion is needed the delay of conversion block will be added to the computation time.

In the second method some changes must take place in the multiplier architecture for reducing the conversion delay and the area of multiplier is unchanged about 2 times but delay of one CSA is added to its computation time. In this method the delay of two propagation adders in the first conversion method is removed and overall its computation time is decreased.

8. References