High-speed modulo \((2^n + 3)\) multipliers

Lei Li\(^{a)}\), Lu Zhou, and Wanting Zhou

Research Institute of Electronic Science and Technology, University of Electronic Science and Technology of China, No.2006, Xiyuan Avenue, High-Tech West Zone, Chengdu 611731, Sichuan, China

\(^{a)}\) leilee@uestc.edu.cn

Abstract: In this express, we propose an improved architecture for modulo \((2^n + 3)\) multiplication on the condition \(n \geq 6\). With this architecture, we can design the fastest among all known modulo \((2^n + 3)\) multipliers. The proposed modulo \((2^n + 3)\) multiplier can improve the state-of-art by 3.2% on the average in terms of area and 10.1% on the average in terms of performance delay.

Keywords: Residue number systems (RNS), multiplier

Classification: Integrated circuits

References


1 Introduction

Residue number systems (RNS) are non-weighted parallel number presentation and operation systems based on Chinese remainder theorem (CRT) and a good alternative to the conventional binary arithmetic. \(\{2^n, 2^n - 1, 2^n + 1\}\) is the most common used moduli set in RNS applications because they have lower implementation difficulty and computational complexity than other moduli forms, when considering the area \(\times\) time\(^2\) product, and also they can offer efficient converters from and to the binary system [1]. The novel moduli set \(\{2^{2n} - 1, 2^n, 2^n + 1\}\) is studied [2]. The 4-modulus base set \(\{2^n - 1, 2^n + 3, 2^n + 1, 2^n - 3\}\) has been proposed [3, 4]. Even though several moduli sets have been proposed using modulo \((2^n + 3)\) operations, the design
of the required modulo \((2^n + 3)\) multipliers is still a key challenge. The improved units for modulo \((2^n + 3)\) multipliers have been proposed [5]. However, they are not very efficient because they employ the conventional method to compute the residue of modulo \((2^n + 3)\). And the execution process of the reference modulo \((2^n + 3)\) multiplier [5] is shown in Section 3.

In this express, we propose an architecture for modulo \((2^n + 3)\) multiplication on the condition \(n \geq 6\), which removes the drawbacks of the state-of-art modulo \((2^n + 3)\) multipliers. With this architecture, we can design the fastest among all known modulo \((2^n + 3)\) multipliers. Static analysis demonstrates the average ratio of efficiency of the proposed architecture to the reference architecture [5] is 1.33. Synthesized results demonstrate the proposed modulo \((2^n + 3)\) multipliers can achieve an average ratio of efficiency of 1.29 against the reference multipliers [5].

2 The feature of modulo \((2^n + 3)\)

For any integer \(X\), there is
\[
\langle X \rangle_{2^n + 3} = x, \quad x \in [0, 2^n + 2]
\]
In other words, the residue set of modulus \((2^n + 3)\) is \([0, 2^n + 2]\).

For modulo \((2^n + 3)\), there are
\[
\langle Z[n] \times 2^n \rangle_{2^n + 3} = \langle -3Z[n] \rangle_{2^n + 3} = \langle 3(Z[n] - 1) \rangle_{2^n + 3} \tag{2}
\]
\[
\langle Z[n - 1 : 0] \times 2^n \rangle_{2^n + 3} = \langle 9Z[n - 1 : 0] \rangle_{2^n + 3} \tag{3}
\]
\[
\langle Z[n - 1 : 0] \times 2^n \rangle_{2^n + 3} = \langle -3Z[n - 1 : 0] \rangle_{2^n + 3} = \langle 3(Z[n - 1 : 0] + 4) \rangle_{2^n + 3} \tag{4}
\]

where \(Z[w : v]\) represents bits of \(Z\) originally located in positions from \(v\) (less significant) to \(w\) (more significant) and the symbol \# is used to concatenate bits.

3 The proposed modulo \((2^n + 3)\) multipliers

Let \(A[n : 0] \times B[n : 0] = P[2n + 1 : 0]\) and the modulo \((2^n + 3)\) multiplication can be described as:
\[
\langle A[n : 0] \times B[n : 0] \rangle_{2^n + 3} = \langle P[2n + 1 : 0] \rangle_{2^n + 3}
\]
\[
= \langle 2^n \cdot P[2n + 1 : 2n] + 2^n \cdot P[2n - 1 : n] + P[n - 1 : 0] \rangle_{2^n + 3} \tag{5}
\]

Using (2)–(4), (5) can be rewritten as:
\[
\langle A[n : 0] \times B[n : 0] \rangle_{2^n + 3} = \langle P[2n + 1 : 0] \rangle_{2^n + 3}
\]
\[
= \left[ \begin{array}{c}
P[2n - 2 : n] \\
\# P[2n - 1] \\
\# + 0 \cdots 0 \\
\# + 0 \cdots 0
\end{array} \right]_{(n-5)\text{ bits}}
\]
\[
+ \left[ \begin{array}{c}
P[2n + 1 : 2n] \\
\# P[2n + 1 : 2n] \\
\# 0
\end{array} \right]_{(n-2)\text{ bits}}
\]
\[
+ \left[ \begin{array}{c}
P[2n - 1] \\
\# 0 + 9
\end{array} \right]_{2^n + 3} \tag{6}
\]
The former 3 terms $\bar{P}[2n - 2 : n] \# P[2n - 1], \bar{P}[2n - 1 : n]$ and $P[n - 1 : 0]$ in (6) can be computed with one $n$-bit CSA (Carry Save Adder) structure at 

\[ (n-5) \text{ bits} \]

first. The remaining three terms $\bar{0} \cdots \bar{0} \# P[2n - 1] \# 0$ and $P[2n + 1 : 2n]$, $\bar{0} \cdots \bar{0} \# P[2n - 1] \# 0$ and 9 are reserved for merging with some other term produced in the latter computation.

\[
\bar{P}[2n - 2 : n] \# P[2n - 1] + \bar{P}[2n - 1 : n] + P[n - 1 : 0] \tag{7}
\]

where $L[n - 1 : 0]$ and $H[n - 1 : 0]$ are sum output data and carry output data of the $n$-bit CSA structure, respectively. And (7) can be rewritten as:

\[
\langle L[n - 1 : 0] + 2H[n - 1 : 0]\rangle_{2^n+3} = \langle L[n - 1 : 0] + H[n - 1 : 0] \# 0\rangle_{2^n+3} \tag{8}
\]

\[
\langle L[n - 1 : 0] + H[n - 2 : 0] \# H[n - 1] \# H[n - 1] - 3\rangle_{2^n+3} \tag{8}
\]

The former 2 terms in (8) can be computed using one $n$-bit binary adder with $L[n - 1 : 0]$ and $H[n - 2 : 0] \# H[n - 1]$ as the two addends and $R[n : 0]$ is the sum of the former two terms in (8). And the third term in (8) just has one bit data in the second bit and is easy to merge with the fourth term and the fifth term in (6). It is reserved for merging the fourth term and the fifth term in (6). The fourth term $-3$ in (8) is very easy to merge with the last term 9 in (6).

\[
L[n - 1 : 0] + H[n - 2 : 0] \# H[n - 1] = R[n : 0] \tag{9}
\]

And (9) can be rewritten as:

\[
\langle R[n : 0]\rangle_{2^n+3} = \langle R[n - 1 : 0] + 0 \cdots 0 \# R[n] \# R[n] - 3\rangle_{2^n+3} \tag{10}
\]

The fourth term, the fifth term in (6) and the third term (8) can be computed with a 1-bit full adder to get the sum $\bar{0} \cdots \bar{0} \# W[4 : 0]$.

\[
\bar{0} \cdots \bar{0} \# P[2n + 1 : 2n] \# 0 \# P[2n + 1 : 2n] + \bar{0} \cdots \bar{0} \# P[2n - 1] \# 0
\]

\[
+ \bar{0} \cdots \bar{0} \# H[n - 1] \# 0 = \bar{0} \cdots \bar{0} \# W[4 : 0] \tag{11}
\]

A 5-bit binary adder is used to compute $\bar{0} \cdots \bar{0} \# R[n] \# R[n] + \bar{0} \cdots \bar{0} \# W[4 : 0]$ and the sum is $\bar{0} \cdots \bar{0} \# G[5 : 0]$. To guarantee from overflow of $\bar{0} \cdots \bar{0} \# G[5 : 0]$,
one extra condition has been added: \( n \geq 6 \).

\[
\begin{align*}
\text{(n-2) bits} & \quad 0 \cdots 0 \not\# R[n] \not\# R[n] + 0 \cdots 0 \not\# W[4 : 0] = 0 \cdots 0 \not\# G[5 : 0] \\
\text{(n-5) bits} & \\
\text{(n-6) bits} & \end{align*}
\]

(12)

One \( n \)-bit binary adder is used to compute \( R[n - 1 : 0] + 0 \cdots 0 \not\# G[5 : 0] \) and \( T[n : 0] \) is the sum:

\[
\begin{align*}
\text{(n-6) bits} & \\
R[n - 1 : 0] + 0 \cdots 0 \not\# G[5 : 0] = T[n : 0]
\end{align*}
\]

(13)

Herein we have to make a simple correction as:

\[
\begin{align*}
\langle T[n : 0] \rangle_{2^n+3} = \left( T[n - 1 : 0] + 0 \cdots 0 \not\# T[n] \not\# T[n] - 3 \right)_{2^n+3}
\end{align*}
\]

(14)

The fourth term \(-3\) in (8), the third term \(-3\) in (10) and the third term \(-3\) in (14) will counteract the sixth third term \(9\) in (6). The final result is \( Y[n : 0] \):

\[
\begin{align*}
\text{(n-2) bits} & \\
T[n - 1 : 0] + 0 \cdots 0 \not\# T[n] \not\# T[n] = Y[n : 0]
\end{align*}
\]

(15)

\( Y[n : 0] \) does not overflow.

Proof: Let us take the worst case, that is \( T[n - 1 : 0] = 2^n - 1 \), and \( T[n] = 1 \). From (15), we get \( Y[n : 0] = 2^n - 1 + 3 = 2^n + 2 \in [0, 2^n + 2] \). Obviously, \( Y[n : 0] \) does not overflow.

Fig. 1 plots the proposed architecture for modulo \((2^n + 3)\) multiplication.

From Fig. 1, it can be obtained the critical path is the timing path that propagates through the \((n + 1) \times (n + 1)\) binary multiplier, the used \( n \)-bit CSA, \( n \)-bit binary adder (1), 5-bit binary adder, \( n \)-bit binary adder (2) and \( n \)-bit binary adder (3).

To explain the execution process of the proposed modulo \((2^n + 3)\) multiplier, we choose \( n = 6 \) as a case study. That is \( 2^6 + 3 = 67 \). Let us consider the case of \( A[6 : 0] = 65 = (1000001)_2 \) and \( B[6 : 0] = 66 = (100010)_2 \). From (6), we have \( P[13 : 0] = 4290 = (1000011000010)_2 \). Further, there are \( L[5 : 0] = 6 = (000110)_2 \), \( H[5 : 0] = 56 = (111000)_2 \) and \( W[4 : 0] = 9 = (001001)_2 \). According, \( R[6 : 0] = 54 = (0110110)_2 \) and \( G[5 : 0] = 12 = (001100)_2 \). Thus, there are \( T[6 : 0] = 66 = (1000010)_2 \) and \( Y[6 : 0] = 2 = (0000010)_2 \).

Under the same inputs, the execution process of the reference modulo \((2^n + 3)\) multiplier \([5]\) is given as follows. \( P[13 : 0] = 4290 = (1000011000010)_2 \) after a \((n + 1) \times (n + 1)\) binary multiplier, as the proposed \((2^n + 3)\) multiplier.

A CSA tree is used to compress 5 terms merged from 6 terms in (6) and it can be obtained that \( L[7 : 0] = 64 = (01000000)_2 \) and \( H[7 : 0] = 36 = (00100100)_2 \). The sum of these 2 terms is \( 136 = (01001000)_2 \). And the final result is \( \langle 136 \rangle_{67} = 2 \).
4 Analysis

The proposed modulo \((2^n + 3)\) multipliers are composed of an \(n\)-bit \(\times\) \(n\)-bit binary multiplier, a \(n\)-bit CSA, a 1-bit full adder, a 5-bit binary adder, three \(n\)-bit binary adders and \((n + 3)\) inverters, while the reference multipliers [5] are composed of 3 CSA, 3 \((n + 1)\)-bit binary adder, 2 \((n + 1)\)-bit \((2:1)\) MUXs and \(n\) inverters.

Since the proposed modulo \((2^n + 3)\) multiplier and the reference modulo \((2^n + 3)\) multiplier [5] both consist of an \((n + 1)\) \(\times\) \((n + 1)\) binary multiplier, the effect of the binary multiplier can be ignored in static analysis. Based on the model [6], we have

\[
T_{Pro,\text{nonMult}} = 2\log_2(n) + \log_2(n-2) + \max\{6, \log_2(n-6)\} + 24 \quad (16)
\]
\[
A_{Pro,\text{nonMult}} = 18n + \frac{3}{2}n \log_2 n + \frac{1}{2}(n - 6) \log_2(n - 6)
+ \frac{1}{2}(n - 2) \log_2(n - 2) + 70 \quad (17)
\]
\[
T_{Ref,\text{nonMult}} = 4\log_2(n+1) + 23 \quad (18)
\]
\[
A_{Ref,\text{nonMult}} = 23n + \frac{9}{2}(n + 1) \log_2(n + 1) + 46 \quad (19)
\]

Static analysis demonstrates the average ratio of efficiency of the proposed architecture to the reference architecture [5] is 1.33.
The proposed modulo \((2^n + 3)\) multipliers in this paper were modeled in structural Verilog HDL for a general value of \(n\) and their operations were exhaustively verified. The proposed modulo \((2^n + 3)\) multipliers in this paper were implemented using TSMC 90 nm CMOS process technology. The Synopsys Design Compiler tool version Y-2006.06-SP4 was used to get the synthesized results. The obtained results are plotted in Fig. 2 and Fig. 3.

![Fig. 2. The delay performance of the proposed multipliers and the reference multipliers [5]](image)

![Fig. 3. The synthesized area of the proposed multipliers and the reference multipliers [5]](image)

From the synthesized results, the proposed modulo \((2^n + 3)\) multipliers can achieve an average delay savings of 10.1% with an average area savings of 3.2% against the reference multipliers [5]. Synthesized results demonstrate that the proposed modulo \((2^n + 3)\) multipliers have a very good performance. Synthesized results demonstrate the proposed modulo \((2^n + 3)\) multipliers can achieve an average ratio of efficiency of 1.29 against the reference multipliers [5], which is shown in Fig. 2 and Fig. 3.
5 Conclusion

In this express, we have proposed an improved architecture for the modulo \((2^n + 3)\) multipliers on the condition \(n \geq 6\). Synthesized results demonstrate that the proposed modulo \((2^n + 3)\) multipliers can achieve an average delay savings of 10.1\% with an average area savings of 3.2\% against the reference multipliers [5].

Acknowledgments

This work was supported by National Natural Science Foundation of China (NSFC) under Grant 61076096 and 61101033.