ROM Table Reduction Techniques For Computing The Squaring Operation Using Modular Arithmetic

Alexander Skavantzos

Louisiana State University, Baton Rouge, Louisiana

Abstract

Modular arithmetic is necessary when using the Residue Number system. Such a system is suitable for high-speed Signal Processing, due to its carry free and parallel nature. In this paper, we present techniques for minimizing the size of the look-up table ROMs used for computing the squaring operation, when arithmetic modulo \( m \) is performed. Such squaring operations are necessary when implementing the quarter squared and the one over eight squared algorithms. These squared-law algorithms are useful in computing multiplicative intensive operations, such as convolutions, correlations and complex multiplications. The presented memory compression schemes can result in significant ROM savings, with the only extra cost of a small overhead.

1. Introduction

A number of important signal processing and communication applications use algorithms which are based on complex operations. Such operations are complex multiplications, linear and cyclic convolutions, autocorrelations and cross-correlations. Since speed and cost are the main design factors, Digital Signal Processing (DSP) engineers and scientists have been looking at various speed/cost efficient techniques for implementing their algorithms.

Modular arithmetic has been quite common in DSP applications, since it can be combined with the use of the Residue Number System (RNS); an integer system capable of performing parallel, carry-free, high-speed arithmetic [1]-[2]. Since the RNS is an integer system, it can easily handle only the simple operations of addition, subtraction and multiplication. Divisions do not always produce integer results and as such, they are not preferred operations in such a system, while magnitude comparisons are difficult because of the unweighted nature of the RNS. As a result, the Residue Number System has been receiving significant attention for Digital Signal Processing applications, where the dominant operations are the operations of addition, subtraction and multiplication.

An RNS is defined by a set of relatively prime integers \( P = \{m_1,...,m_L\} \), where \( P \) is called the set of moduli. In such a system, arithmetic is performed "modulo \( m_i \)", while the computations between two different moduli \( m_i \) and \( m_j \) are independent from each other. Since RNS implementations of DSP tasks rely on additions, subtractions and multiplications, the focus of many RNS researchers has been on efficient implementations of the above arithmetic operations "modulo \( m \)". References [3]-[4] represent some of the existing work on addition and subtraction in RNS. On the other hand, multipliers "modulo \( m \)" can be implemented using several techniques. Among these techniques, one can refer to the look-up table technique using ROMs [5]-[6], the index calculus technique which can only be used with prime moduli [7] and the quarter squared algorithm technique [8]-[10].

Recently, a new squared law algorithm has been developed for modulo \( m \) arithmetic. It is "the one over eight squared algorithm" and can be used in computing convolutions and correlations [11].

The following points should be clarified here.

1. Both the quarter squared and the one over eight squared algorithms can be used with arithmetic modulo \( m \), for any modulus \( m \).
2. These two algorithms require only squaring operations and additions, but not two-operand multiplications.
3. When the two algorithms are used with arithmetic modulo \( m \), ROM look-up tables can be used to perform the squaring operations modulo \( m \).
4. Both the algorithms reduce the size of the ROM tables from \( 2^{2n} \) words to \( 2^n \) words, where \( n \) is the wordlength used.

This paper presents techniques for minimizing
the size of the look-up ROMs used for computing the useful squaring operations, when arithmetic modulo \( m \) is performed. Designs for the cases of \( m=2^n \) and \( m=2^n-1 \) are presented in the paper. The case of \( m=2^n+1 \) is a simple modification of the case of \( m=2^n-1 \). The techniques can be modified to apply to any general modulus \( m=2^k-k \), but in this case the extra overhead increases.

2. The Memory Compression Schemes

We first present the memory compression schemes when the modulus used is \( m=2^n \). Consider a number \( A \) belonging in \( \mathbb{Z}_{2^n} = \{0,1,\ldots,2^n-1\} \). Then, \( A \) has an \( n \)-bit binary representation as in

\[
A = a_{n-1}a_{n-2}\cdots a_1a_0
\]

(1)

Our task is to compute \( \langle A^2 \rangle_{2^n} \), where \( \langle \cdot \rangle_m \) denotes the operation \( \cdot \) modulo \( m \). If the table look-up approach is used, then a ROM table of size \( 2^n \times n \) bits needs to be used, before any compression is applied.

Alternatively, consider the decomposition of \( A \) into a 1-bit upper byte \( A_{H1} = a_{n-1} \) and an \((n-1)\)-bit lower byte \( A_{L1} = a_{n-2} \cdots a_0 \). Then

\[
A = A_{H1}2^{n-1} + A_{L1}
\]

(2)

while

\[
A^2 = A_{H1}^2 2^{2n-2} + A_{H1}A_{L1} 2^n + A_{L1}^2
\]

(3)

and for \( n>2 \) one gets

\[
\langle A^2 \rangle_{2^n} = \langle A_{L1}^2 \rangle_{2^n}
\]

(4)

According to (4) the desired square can be computed using a ROM table of size \( 2^{n-1} \times n \) bits, or 50% reduction in the size of the ROM has been achieved without any extra overhead.

Consider now decomposing \( A \) into a 2-bit upper byte \( A_{H2} = a_{n-1}a_{n-2} \) and an \((n-2)\)-bit lower byte \( A_{L2} = a_{n-3} \cdots a_0 \). We then get

\[
A = A_{H2} 2^{n-2} + A_{L2}
\]

(5)

or

\[
A^2 = A_{H2}^2 2^{2n-4} + A_{H2}A_{L2} 2^{n-1} + A_{L2}^2
\]

(6)

while for \( n>4 \) we get

\[
\langle A^2 \rangle_{2^n} = \langle A_{H2}A_{L2} 2^{n-1} + A_{L2}^2 \rangle_{2^n}
\]

(7)

The amount \( \langle A_{H2}A_{L2} 2^{n-1} \rangle_{2^n} \) is given below.

\[
\begin{align*}
\langle A_{H2}A_{L2} 2^{n-1} \rangle_{2^n} &= \\
0 &\quad \text{if } a_{n-2} = 0 \\
\langle A_{L2} 2^{n-1} \rangle_{2^n} &\quad \text{if } a_{n-2} = 1
\end{align*}
\]

(8)

Since

\[
\langle A_{L2} 2^{n-1} \rangle_{2^n} = \langle (a_{n-3}a_{n-4}\cdots a_0)2^{n-1} \rangle_{2^n} = \\
\langle (a_{n-3}a_{n-4}\cdots a_1)2^n + a_0 2^{n-1} \rangle_{2^n} = a_0 2^{n-1},
\]

then (7) and (8) give

\[
\langle A^2 \rangle_{2^n} = \\
\begin{align*}
\langle A_{L2}^2 \rangle_{2^n} &\quad \text{if } a_{n-2} = 0 \\
\langle a_0 2^{n-1} + A_{L2}^2 \rangle_{2^n} &\quad \text{if } a_{n-2} = 1
\end{align*}
\]

(9)

Suppose that \( \langle A_{L2}^2 \rangle_{2^n} \) is represented in binary as \( \langle A_{L2}^2 \rangle_{2^n} = b_{n-1}b_{n-2}\cdots b_1b_0 \). Then (9) gives

\[
\langle A^2 \rangle_{2^n} = \\
\begin{align*}
\langle A_{L2}^2 \rangle_{2^n} &\quad \text{if } a_{n-2} = 0 \\
\langle a_0 2^{n-1} + A_{L2}^2 \rangle_{2^n} &\quad \text{if } a_{n-2} = 1
\end{align*}
\]

(10)

where

\[
c_{n-1} = a_0 \oplus b_{n-1}
\]

(11)

and where \( \oplus \) denotes the EXCLUSIVE-OR operation.

Figure 1 shows the computation of \( \langle A^2 \rangle_{2^n} \) using a \( 2^{n-2} \times n \) ROM and the extra overhead of one exclusive-or gate and a 2-to-1 multiplexer. In this case, a 75% reduction in the size of the ROM has been achieved.

The computation of \( \langle A^2 \rangle_{2^n} \) based on decomposing \( A \) into a 3-bit upper byte \( A_{H3} = a_{n-1}a_{n-2}a_{n-3} \) and an \((n-3)\)-bit lower byte \( A_{L3} = a_{n-4} \cdots a_0 \) is shown in Figure 2. The derivations are not shown due to restrictions in the size of the paper. Figure 2 demonstrates that the desired square can be computed using a ROM of size \( 2^{n-3} \times n \) bits plus the extra overhead of few gates and a 4-to-1 multiplexer. An 87.5% savings in ROM bits has been achieved in this case.

We now present the memory compression schemes for arithmetic modulo \( 2^n-1 \). Our number \( A \) that needs to be squared is belonging in \( \mathbb{Z}_{2^n-1} = \{0,1,\ldots,2^n-2\} \) and thus \( A \) assumes an \( n \)-bit binary representation as in (1). In our effort to
minimize the size of the paper we will not show derivations. Figures 3 and 4 show the obtained result.

Figure 3 shows the computation of \( <A^2>_{2^{n-1}} \) based on decomposing \( A \) into a 1-bit upper byte \( A_{U1} = a_{n-1} \) and an \( (n-1) \)-bit lower byte \( A_{L1} = a_{n-2} \ldots a_0 \). It relies on a ROM of size \( 2^{n-1} \times n \) bits instead of \( 2^n \times n \) bits that would have otherwise been required. The extra overhead is a modulo \( 2^{n-1} \) adder and a 2-to-1 multiplexer. Figure 4 shows the computation of \( <A^2>_{2^m} \) based on a decomposition of \( A \) into a 2-bit upper byte \( A_{U2} = a_{n-1}a_{n-2} \) and an \( (n-2) \)-bit lower byte \( A_{L2} = a_{n-3} \ldots a_0 \). Here the required ROM is of size \( 2^{n-2} \times n \) bits and the extra overhead two modulo \( 2^{n-1} \) adders and a 4-to-1 multiplexer.

3. Conclusion

We have presented memory compression schemes for minimizing the size of the table look-up ROMs used for computing the squaring operation. These techniques resulted in significant savings in ROM bits at an extra cost of a small overhead. The presentation considered modulo \( 2^n \) and modulo \( 2^{n-1} \) arithmetic. Simple modifications of the case of arithmetic modulo \( 2^{n-1} \) can result in arithmetic modulo \( 2^n+1 \). For example, by replacing the ROM and the adder of Figure 3 with a ROM and an adder performing arithmetic modulo \( 2^n+1 \) and by negating modulo \( 2^n+1 \) the left input to the adder, the scheme of Figure 3 would then be computing \( <A^2>_{2^n+1} \) for any \( A < 2^n \). The presented techniques can be modified to apply to any general modulus \( m = 2^n-k \), but in this case the extra overhead increases. The overall result is that efficient implementations of the quarter squared and the one over eight squared algorithms can be obtained, using reduced address space ROM tables.

Acknowledgment

This work has been supported in part by NSF Grant No. CCR8910187.

References


Figure 1: The computation of $<A^2>_2^n$ based on a $2^{n-2} \times n$ ROM.

Figure 2: The computation of $<A^2>_2^n$ based on a $2^{n-3} \times n$ ROM.
Figure 3: The computation of $<A^2>_{2^{n-1}}$ based on a $2^{n-1} \times n$ ROM.

Figure 4: The computation of $<A^2>_{2^{n-1}}$ based on a $2^{n-2} \times n$ ROM.