Analyzing Methods for Truncated Binary Multiplication for FIR Filter Design

MADANU RAMYA\textsuperscript{1}, CH. UMA SHANKAR\textsuperscript{2}

\textsuperscript{1}PG Scholar, Dept of ECE (VLSI ID), Universal College of Engineering and Technology, Guntur, AP, India,
Email: madanuramya@gmail.com.
\textsuperscript{2}Assistante Professor, Dept of ECE, Universal College of Engineering and Technology, Guntur, AP, India,
Email: sankarch.4u@gmail.com.

Abstract: Multiplication operations are frequently required in digital signal processing. To increase the speed with which these are done, parallel multipliers can be used. These however require a large area on the chip and consume much power. An important goal would therefore be to reduce the area and delay requirements. Proposed truncated multipliers overcome the previously proposed truncated multipliers since provide lower error, lower power dissipation, lower area occupation and also provide higher working frequency. The circuits are also easily implemented and allow an automatic HDL description as a function of bit width and desired error. Multiple constant multiplication/accumulation in a direct FIR structure is implemented using an improved version of truncated multipliers.

Keywords: Finite Impulse Response (FIR), Digital Signal Processing (DSP), Truncated Multipliers.

I. INTRODUCTION

Finite impulse response (FIR) digital filter is one of the fundamental components in many digital signal processing (DSP) and communication systems. It is also widely used in many portable applications with limited area and Power budget. A general FIR filter of order \( M \) can be expressed as

\[
y[n] = \sum_{i=0}^{M-1} a_i x[n - i].
\]

(1)

In case of linear phase, the coefficients are either symmetric or anti symmetric with \( a_i = a_{M-i} \) or \( a_i = -a_{M-i} \). There are two basic FIR structures, direct form and transposed form, as shown in Fig. 1 for a linear-phase even-order FIR filter. In the direct form in Fig. 1(a), the multiple constant multiplication (MCM)/accumulation (MCMA) module performs the concurrent multiplications of individual delayed signals and respective filter coefficients, followed by accumulation of all the products. Thus, the operands of the multipliers in MCMA are delayed input signals \( x[n - i] \) and coefficients \( a_i \). In the transposed form in Fig. 1(b), the operands of the multipliers in the MCM module are the current input signal \( x[n] \) and coefficients. The results of individual constant multiplications go through structure adders (SAs) and delay elements.

In the past decades, there are many papers on the designs and implementations of low-cost or high-speed FIR filters \cite{1}–\cite{13}, \cite{15}–\cite{19}. In order to avoid costly multipliers, most prior hardware implementations of digital FIR filters can be divided into two categories: multiplier less based and memory based. A weighted sum of partial products. When an application requires a multiplication output with \( n \) bits

Fig.1. Structures of linear phase even-order FIR filters: (a) Direct form and (b) transposed form.
Multiplication involves two basic operations, the generation of the partial products and their sum, performed using two kinds of multiplication algorithms, serial and parallel. Serial multiplication algorithms use sequential circuits with feedbacks: inner products are sequentially produced and computed. Parallel multiplication algorithms often use combinational circuits and do not contain feedback structures. A full-width digital \( n \times n \) multiplier computes the \( 2n \) bits output, as, it is substituted by a truncated multiplier, which is a \( n \times n \) multiplier with \( n \) bits output.

In this chapter, an introduction to the problem, the tools used, the methods, and the Structure of the report and the chapter consists of 2 sub-chapters:

- The problem – an introduction to the problem and what to be done
- The project – Tools and hardware, methods and the structure of the report.

A. The Problem

Problem Description: Truncation is a well-known method to reduce the hardware area of multipliers. By applying truncation significant hardware savings can be achieved at the cost of precision. The problem to be solved in this project is to analyze several methods and degrees of performing truncated multiplication and to determine which method/degree would be most efficient for 8 bits.

There are Two Main Objectives:

- To determine the error for several methods and degrees of truncation for 4 and 8.
- To design and implement the parallel multiplier in order to observe the reduction in area and to confirm if the method works.

1.2 Xilinx ISE 9.2i: Xilinx ISE has many uses but in this project it is used to synthesize models made in ModelSim. It simulates exactly how a model works on a specific device and gives very detailed analytical information, such as timings, RTL schematics and hardware utilization on the device.

II. THEORETICAL BACKGROUNDS

A. Binary Multiplication

An introduction to the pen-and-paper method of performing binary multiplication is briefly given below. Fig. 2 shows the process of multiplying two binary numbers, the multiplicand and the multiplier. Notice that the width of the product is twice that of the inputs. The green box outlines the partial product matrix and the red box outlines a single partial product. The first step in this method is to form the partial product matrix in which each element is obtained by AND’ing the appropriate bits from the multiplicand and the multiplier.

Another way to look at this is:

- If the multiplier bit is 0, the partial product is also 0,
- If the multiplier bit is 1, the partial product is equal to the multiplicand,
- Repeat for every multiplier bit.

Notice that this gives a number of partial products equal to the width of the multiplier. To obtain the final product the elements in the columns (from right to left) are added using binary logic. Any carries are carried on to the next column. The result of this operation is stored in one bit of the product and the operation is repeated for each remaining column.

Rounding: Conventionally an \( n \)-bit multiplicand and an \( n \)-bit multiplier would render a \( 2n \)-bit product. Sometimes an \( n \)-bit output is desired to reduce the number of stored bits. This is done in two steps.

- When rounding to \( n \)-bits a ‘1’ is added to the \( n-1 \)’th bit and a potential carry is propagated to the \( n \)’th bit and so on.
- The unwanted bits (the \( n-1 \)’th least significant bits) are discarded.

In Fig. 2 rounding down to 4 bits corresponds to adding a ‘1’ to \( p_3 \) and then discarding the 4 least significant bits. Rounding in decimal numbers is done in a similar way, that is if you add a ‘5’ instead of a ‘1’ to the \( n-1 \)’th bit. This will not work if the product corresponds to a number unless the bits discarded are fractional bits. To give an example in decimal numbers: 213.456 rounded to 3 decimals would be 213 which is correct, while 213.456 rounded to 13 would result in a very different number.

![Fig.2. 4x4 bit binary multiplication with truncation degree T=3.](image)

![Fig.3. 8x8 bit binary multiplication of truncated degree t=8.](image)

Truncation: Truncation is a method where the least significant columns in the partial product matrix are not formed. The amount of columns not formed in this way, \( T \), defines the degree of truncation and the \( T \) least significant bits of the product always result in ‘0’. The algorithm behind
Analyzing Methods for Truncated Binary Multiplication for FIR Filter Design

truncated multiplication is the same as when dealing with non truncated multiplication regardless of the truncation degree. The effect is illustrated in Fig.2, where a truncation degree of $T = 3$, is applied. Notice that the columns to the right of the maroon vertical line are missing. A loss of precision follows as a result of truncation since the least significant bits of the resulting product are always ‘0’. Notice that the truncated product is always equal to or lower than the real product since ‘1’s are converted to ‘0’s and never the other way around as described in Fig.3. This error, however, can be compensated for by introducing a correction constant and thus leading to a truncation error that is acceptable in most cases when dealing with digital signal processing. In this report the correction constant is found by simulating the multiplier (without correction) and the average precision error over 1,000,000 million samples is then used in a second simulation run as the correction constant.

III. MULTIPLIERS

Essentially there are two types of multipliers. In Fig.4 a sequential multiplier is shown in the left and a parallel multiplier is shown on the right. The general idea of a sequential multiplier is to use the same components repeatedly during many clock cycles whereas parallel multipliers use many components within just one clock cycle. Essentially, parallel multipliers provide speed, while sequential multipliers provide area efficiency. Since the scope of this project is to achieve hardware savings of parallel multipliers, sequential multipliers will not be considered.

A. Full-width Multiplier

A Full-width digital $n \times n$ multiplier computes the $2n$ bits output as a weighted sum of partial products. Parallel-tree multipliers reduce the matrix of partial products to two rows using a combination of full-adders and half-adders. The remaining two rows are then added by a carry propagate adder to produce the final result. In this section at first the most common used techniques, Wallace and Dadda trees, are described; then details on an efficient VLSI implementation.

Wallace-Tree Multiplier: The reduction scheme published by Wallace begins by grouping the partial-product matrix into sets of three rows. Each set is reduced to two rows using half-adders on sets of two bits and full-adders on sets of three bits. Excess rows that do not belong to a set of three are passed to the next reduction stage unmodified. Each reduction stage is processed in a similar way until only two rows remain. Then a final CPA (Carry Propagate Adder) is used. Fig.5 shows the dot diagram illustrating Wallace reduction for an $8 \times 8$ multiplier. Dot diagrams, developed by Dadda, are a convenient means for visualizing the placement of full-adders and half-adders. In such diagrams, dots represent bits, given by partial products. For example, the upper-right dot in the multiplication matrix is $x_8 y_8$. The circle with three dots represents a Full-Adder (FA). In the next stage the circle is substituted by its outputs: a bit sum, a dot in the same column, and a bit carry, a dot in the column on the left. Same reasoning is valid for the half-adder, represented by a circle with dashed line. Summarizing in the Wallace tree the number of the operands is reduced at the earliest opportunity, so, if there are $m$ dots in the column we immediately apply $\lfloor m=3 \rfloor$ full adder to that column. This tends to minimize the overall delay by making the final CPA as short as possible.

Dadda-Tree Multiplier: Wallace’s method is refined by Dadda [20] into a technique that is optimum in the sense that it uses a minimum number of full-adders. At each stage of

---

**Fig.4.** left: a sequential Multiplier, right: a parallel multiplier sources.

**Fig.5.** Wallace reduction for 8x8 multiplier.
reduction, the height $h$ of the matrix of Partial Products can be reduced following figure. Dadda’s reduction uses the full-adder and the half-adder only if the height of the matrix can be reduced following Dadda series, until 2 rows are reached. Fig. 6 shows Dadda reduction for an $8 \times 8$ multiplier. The partial-product matrix has a maximum height of 8, so the first step of reduction leads to $h = 6$, then 4; 3; 2; four reduction steps are necessary.

Array Multiplier: Array multiplier is a very common type of parallel multiplier. Array multipliers are not as fast as tree multipliers, but their regular structure and local interconnect are advantages for very-large-scale-integration (VLSI) implementation. ARE AND gates, which generate the unsigned partial-product bits as described as shown in Fig.7.

B. Truncated Multipliers

A truncated multiplier is an $n \times n$ multiplier with $n$ bits output. As it is shown if Fig. 8 the partial products can be divided into two subsets. The least significant part (LSP) includes the $n$ less significant columns of the partial product matrix, while the most significant part (MSP) includes the remaining columns. The full-width multiplier output, $P$ is given by

$$ P = SMSP + SLSP \tag{2} $$

Where SMSP and SLSP represent the weighted sum of the elements of MSP and LSP respectively.

When a $n$ bits output is needed, the most accurate choice is using the full rounded multiplier: it computes all the matrix of partial products, add a constant to the result on $2n$ bits and takes only the first $n$ bits of the sum. The error introduced by the full rounded multiplier is calculated in Sec. 1.3.1. Unfortunately the full rounded multiplier is the solution with the highest area occupation and power dissipation. Second possibility is using a truncated multiplier in which the partial products of the LSP are discarded assuming that their contribution to the $n$ most significant bits of the output is negligible. This solution is very advantageous in terms of hardware performances. For example in

Fig.8 subdivision of the matrix of partial products for unsigned multiplier $n=8$.

Fig.8 there is the implementation of the array truncated multiplier with $n = 8$. The cells for the LSP matrix are not present and the final circuit halves the number of cells compared to the full-width one. However, a straightforward analysis shows that the direct elimination of the partial products of the LSP causes a very big error
Fig. 9. truncated multiplier.

Fig. 8 there is the implementation of the array truncated multiplier with \( n = 8 \). The cells for the LSP matrix are not present and the final circuit halves the number of cells compared to the full-width one. However, a straightforward analysis shows that the direct elimination of the partial products of the LSP causes a very big error bounded by \((n/2-1)\) lsb, where lsb is the weight of the least significant bit of the result. Fig. 9 shows the truncated reduction of 8x8 multiplier. It needs very less number of half adders and full adders.

IV. VHDL IMPLEMENTATION

The VHDL implementation, however, is much more interesting to have a closer look at a top view design has been given in the design chapter (4.2) but in this chapter we will go into details as to how exactly the different components were implemented. This is what 0 is all about. Finally it is explained how the above mentioned models have been tested to confirm that they are in fact correct and/or reliable. The parameter used to define the level of truncation is different in the VHDL implementation. Simulation link uses the parameter \( k \) where \( n + k \) is the number of columns used from the partial product matrix while truncating. Contrarily, the VHDL implementation uses the parameter \( T \) which is defined as the number of columns not used from the partial product matrix. In the beginning of the project \( k \) was used but this was a poor choice and instead \( T \) was used when the VHDL model was implemented as it seemed more logical.

V. RESULTS AND TIMING DIAGRAMS

Fig. 10 shows the timing diagram of 4x4 multiplier .here last four bits of the output is truncated and rounded and getting the less error. Fig. 11 shows the timing diagram of 4x4 multiplier .here last four bits of the output is truncated and rounded and getting the less error.

VI. CONCLUSION

The Multiplier was also implemented and simulated in ModelSim using the hardware description language VHDL. Furthermore the VHDL model was synthesized using Xilinx ISE to evaluate hardware savings. To conclude the project it can be said that even high levels of truncation lead to a relatively small precision error. This means that truncated multipliers offer significant hardware savings for applications, which do not require exact multiplication. Given specific hardware and precision constraints, the right number of columns to truncate for 4 and 8 bits can easily be determined. Future work could be to extend to multiple multipliers and/or multipliers that form the partial product matrix through other.

VII. REFERENCES


