Microelectronics Journal 45 (2014) 775–780

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

A comparative study of energy/power consumption in parallel decimal multipliers

Amin Malekpour [a], Alireza Ejlali [a,*], Saeid Gorgin [b]

[a] Sharif University of Technology, Computer Engineering, Embedded Systems Research Laboratory (ESRLab), Azadi Ave, Tehran, Iran
[b] Institute for Research in Fundamental Sciences (IPM), School of Computer Science, Iran

ARTICLE INFO

Article history:
Received 19 April 2013
Received in revised form
16 February 2014
Accepted 18 February 2014
Available online 4 April 2014

Keywords:
Decimal multiplication
Parallel multiplier
Energy consumption
Power consumption

ABSTRACT

Decimal multiplication is a frequent operation with inherent complexity in implementation. Commercial and financial applications require working with decimal numbers while it has been shown that if we convert decimal number to binary ones, this will negatively influence the preciseness required for these applications. Existing research works on parallel decimal multipliers have mainly focused on latency and area as two major factors to be improved. However, energy/power consumption is another prominent issue in today's digital systems. While the energy consumption of parallel decimal multipliers has not been addressed in previous works, in this paper we present a comparative study of parallel decimal multipliers, considering energy/power consumption, leakage and dynamic power consumption, beside latency and area. This study can provide some guidelines for EDA tools and hardware designers to choose proper multiplier based on given applications and design constraints. All designs in were implemented using VHDL and synthesized in Design-Compiler toolbox with TSMC 45 nm technology file.

1. Introduction

Due to the dramatic advances in VLSI technology, hardware realization of many complex functions is possible now. There is an intensive request for decimal computations in various critical applications such as scientific, commercial, financial and internet-based applications [1]. Since binary representation cannot ensure preciseness for simple decimal fractions (e.g. 0.2), design and implementation of decimal arithmetic algorithms draws attention of scientists and practitioners.

On account of the importance of decimal arithmetic, in recent years IEEE has added new feature of decimal to IEEE 754 Standard for Floating-Point Arithmetic (IEEE 754–2008) [2]. Decimal multiplication is one of the widely used operations in throughout the computation systems [3].

It has been observed that the considerably wider digit set of decimal numbers has made decimal multiplication more complicated than binary multiplication [4]. Indeed, decimal multiplication is a complex, time consuming, power hungry, and high frequent operation. Decimal multiplication has been realized by iterative hardware algorithms in some recent announced processors (e.g. IBM POWER6 [5], IBM z10 [6] and IBM z900 [7]), and the area consumption of an iterative multiplier is lower as compared to parallel implementations. However, there are several promising points for exploiting parallel multipliers especially because the latency of a parallel multiplier makes it more appealing for decimal intensive calculations.

So far designers of conventional arithmetic modules have mainly focused on latency and area as design parameters to be improved but power/energy consumption is another important key factor because of limited power budget in embedded or portable systems and the issue of heat generation by fast, complex and power hungry circuits [8,10].

The algorithms that have been proposed for decimal multiplications mostly iterate through the multiplier and add the corresponding multiplicand digits to a register successively. But, there are various architectures proposed for parallel decimal multiplication, as will be described in the next section; however they have only focused on latency and area and did not study the energy/power consumptions.

In this paper we provide a comparative study so that designers can identify an appropriate parallel decimal multiplier among the existing ones which can meet their constraints, rather than proposing a new design for such multipliers. It has been observed that comparative studies (like the works in [11–16]), where only existing designs are studied and compared (without providing any new design), provide useful information and guidelines to designers and researchers, and we aim at providing a similar research for parallel

* Correspondence to: Sharif University of Technology, Department of Computer Engineering, P.O. Box 11155-9517, Tehran, Iran. Tel.: +98 21 66166621; fax: +98 21 66019246.
E-mail addresses: malekpour@ce.sharif.edu (A. Malekpour), ejlali@sharif.edu (A. Ejlali), gorgin@ipm.ir (S. Gorgin).

http://dx.doi.org/10.1016/j.mejo.2014.02.014
0026-2692/

decimal multipliers. To do this, we compare seven parallel decimal multipliers (indeed almost all the best designs that have been so far presented by the researchers) from the viewpoint of energy/power consumption, leakage and dynamic power consumption as well as latency and area. Based on this study, we will propose some guidelines which could help EDA tools and hardware designers to use an appropriate multiplier for a given application and design constraints.

Overall, we make the following contributions:

- A comparative study of parallel decimal multipliers is presented and the advantages and disadvantages of each design have been analyzed. To the best of our knowledge, we are not aware of any published papers that include the energy/power consumption results of parallel decimal multipliers.
- We propose some guidelines for EDA tools and hardware designers based on our comparative study.
- We measure and report power consumption of parallel decimal multipliers for the first time and show some designs consume large power to gain performance.

The rest of this paper is organized as follows. In Section 2, we briefly describe the best seven parallel decimal multipliers. A theoretical foundation of leakage and dynamic power, in Section 3, paves the way for the discussion on the energy consumption and analysis of experimental results in Section 4. In Section 4 we also present comparisons from different viewpoints, our flow of implementation and synthesis of designs, and a number of guidelines for selecting suitable parallel decimal multiplier for a given application. Finally, conclusions and prospects for further research are found in Section 5.

2. Decimal multipliers

The algorithm of decimal multiplication is composed of three steps: partial product generation (PPG), partial product reduction (PPR), and redundant to non-redundant conversion. The PPG module generates matrix of partial products consisting of the BCD and redundant digits. This matrix is sent as input to PPR. In parallel decimal multipliers, partial products are compressed to a single depth redundant decimal or a double BCD digits format via a reduction tree. Finally, a redundant to non-redundant decimal converter or a decimal adder provides final product result. Researchers have proposed various techniques and algorithms for implementing these parts that we will describe in this section briefly.

The first implementation of parallel decimal multiplier is proposed in [19]. In this design, in the PPG part for providing X–$9X$ multiples (where X is multiplicand), only multiples $2X$, $5X$ and $10X$ are generated and a 9′s complement module has been used to compute $-X$ and $-2X$. Then two multiplexer have been used to select appropriate multiples, which are added by means of a 3:2 radix-10 carry-save adders (CSA), based on value of recoded multiplier digits. The radix-10 CSA reduces two BCD digits and one input carry bit to one BCD digit and one carry bit.

In the PPR part of [19], radix-10 CSA and counters are used to add a carry save operands with BCD operands in a 6-level tree. In the first level of the PPR part, eight CSAs have been used and the carries of other partial products are added by means of a carry-counter (CC). A carry-counter adds eight carries of the same weight and produces a BCD digit. Finally, the radix-10 carry-save representation is converted into the BCD by means of a radix-10 carry save to BCD converter, in which the inputs are the $2n$ digit/bit (outputs of the PPR part) and the output is 32 digit BCD.

For fast partial product generation and reduction, a novel algorithm is proposed in [20] that uses 4221 and 5211 encodings. In PPG part multiples of $-2X$, $-X$, X, $2X$, $5X$, and $10X$ are produced by shift and encoding conversion. In this design, carry-save (4221) encoding is utilized via proposing a new decimal 3:2 carry-save adder (CSA) that reduces three input digits to equally weighted carry and sum, where all inputs and outputs digits are represented in (4221) encoding. With eight levels of 3:2 CSA modules, in the PPR part, a 32:2 decimal CSA is constructed. The last step in this multiplier requires a 32-digit BCD (128 bits) adder. This adder is a 32-digit conditional speculative adder, which provides input carry of each digit via a quaternary tree.

Negative multiples cause some area and delay inefficiency in [19,20]. Therefore, in [22], the unsigned multiples of multiplicand X, $2X$, $4X$, and $5X$ are used for partial product generation. For partial product reduction, two alternative reduction schemes are proposed: delay-optimized and area-optimized. The former scheme, which is called [22]-a in this paper, uses a novel carry-free adder (ODDS adder) and its restricted input varieties to achieve a VLSI-friendly recursive partial product reduction tree, while the latter scheme, which is called [22]-b, employs binary 4:2 compressors augmented with $+0/6$ correction logic. Finally in the last step, for both architectures, a redundant to non-redundant converter is used to convert ODDS products to BCD digits.

Although, the PPG schema of [22] does not generate negative multiples, however, generation of $4X$, in respect to other multiples, takes two times delay. Hence, a new PPG method is presented in [21], which only generates $2X$, $5X$, $(8X_l+8X_h)$, $(9X_l+9X_h)$ to compose other multiples. This new method avoids $4X$ generation and negation logic at the cost of increase in hardware complexity of the PPG in order to generate $8X$ and $9X$.

In order to reduce the partial product in [21], a 6-level reduction tree is used. The carry input of BCD-FAs in the first level of the PPR does not use. Only half of the generated carries in the PPR levels are absorbed by BCD-FAs. In order to convert the unused carries to BCD digits, a 9:4 counter and a 6:3 counter are used in this design. For more acceleration a speculation logic is used and the carryout of the very last BCD-FAs is routed to a multiplexer that selects between b and $b+1$. Actually, the increment operation in speculation logic is performed off the critical path and just takes two logic levels. The last step in this multiplier requires a 32-digit BCD (128 bits) adder. This adder is a 32-digit conditional speculative adder, which provides input carry of each digit via a quaternary tree.

Architectures of two parallel decimal multipliers are presented in [23], which are improved designs of multiplier in [20]. In the PPG part of 16-digit radix-5 architecture multiples $-X$, $2X$, $-2X$, $5X$, and $10X$ are generated. Then 32 decimal partial products are generated, half coded in (4221) and the other half coded in (5211). In order to reduce this aligned partial product, mixed (4221/5211) decimal digit CSA trees have been used.

In the PPG part of 16-digit radix-10 architecture, multiples X, $2X$, $3X$, $4X$, and $5X$ coded in (4221) and 17 partial products are generated. In order to generate the 17 partial products, an encoding of the multiplier into 16 SD radix-10 digits is needed. When the sign of the corresponding SD radix-10 digit is negative, a level of XOR gate inverts the outputs of the multiplexer. Then the partial products, coded in (4221), are aligned and reduction is performed by means of decimal digit CSA trees.

For redundant to non-redundant converting, both architectures have used a quaternary tree adder according to conditional speculative decimal addition proposed by the same authors in [24].

It is worthwhile to note that the primary concerns of all aforementioned works are latency and area – they do not report any energy/power consumption of parallel decimal multipliers.

3. Energy/power consumption

As transistor counts and clock frequencies have increased, power consumption has skyrocketed and now is an important design constraint [25].