Ernest JAMRO[1,2], **Maciej WIELGOSZ**[1,2], **Kazimierz WIATR**[1,2]

Akademia Górniczo-Hutnicza, Kat. Elektroniki (1), Akademickie Centrum Komputerowe Cyfronet (2)

Novel Reduced-Width Multiplier Structure Dedicated for FPGAs

Streszczenie. *Niniejszy artykuł prezentuje nową strukturę układu mnożącego o skróconej szerokości z dodatkowym układem kompensacji błędu odcięcia. W przeciwieństwie do prezentowanych dotąd technik kompensacji błędu odcięcia, prezentowana architektura jest dedykowana dla układów programowalnych FPGA i nie wymaga dodatkowych zasobów logicznych a mimo to umożliwia znaczącą redukcję błędu. (**Nowa struktura układu mnożącego o skróconej szerokości przeznaczona dla układów FPGA**)*

Abstract. *This paper describes a novel structure of reduced-width multiplier. The main idea is to use a special architecture to compensate for the truncation error. The architecture is dedicated to FPGAs (Filed Programmable Gate Arrays) and does not require any additional FPGAs resources in comparison to the direct truncation.*

Słowa kluczowe: układy programowalne FPGA, układ mnożący.
Keywords: FPGA (Field Programmable Gate Aray), multiplier.

Introduction

Multiplication is a fundamental arithmetic operation therefore the amount of resources occupied by this operation is crucial for a whole system. Multiplication is usually carried out with full bit-width, i.e. the output bit-width is the sum of inputs bit-widths. For example if both inputs bit-widths equal n, the output bit-width equals $2 \cdot n$. In this case no calculation error is generated. Nevertheless, in most real-word cases, the input bit-width n is the same as the output bit-width. An example is a filter or floating-point multiplier. In these cases, usually the n-LSBs (Least Significant Bits) of the multiplication result are simply ignored.

Therefore the main idea behind this paper is to carry-out multiplication is such a way that the n (or generally n-w) LSBs of the results are ignored during the multiplication process. Consequently, the multiplier area is reduced by the cost of an additional calculation error. Another way of decreasing calculation error is to add additional error compensation logic as in e.g. [1, 2]. These publications also discussed thoroughly other (previously published) error compensation methods. Nevertheless, these error compensation methods are dedicated to ASIC (Application Specific Integrated Circuit) and are not optimal for FPGA [3] designs. Therefore the main objective of this paper is to develop a similar error compensation method which is suitable for FPGAs.

In this paper an assumption is made that multipliers are implemented in Configurable Logic Blocks (CLBs) [3]. However, dedicated multipliers incorporated in DSP blocks [3] are now available in FPGAs and often they are preferable to CLB-type of multipliers. Nevertheless, according to authors' experience and the publication [4], using dedicated multipliers complicates routing and often leads to additional propagation delays as they are located in the particular FPGAs sites. Besides, the number of dedicated multipliers is limited for a given FPGA, and in certain cases a combination of dedicated and CLB-type multipliers can increase design functionality. Similarly a combination of CLB-type and dedicated multipliers can be used within a single wide-bits multiplier, e.g. in a 24×24-bit multiplier when 18×18-bit built-in multipliers are available. This combination is especially recommended for reduced-width multipliers.

ASIC Implementation

One of a fundamental multiplier architecture for ASIC is parallel-array multiplier [5], which block diagram is presented in Fig. 1. It should be noted that this multiplier architecture is not optimal for ASIC [5], nevertheless it is enumerated hereby (similarly as in [1, 2]) because of its simplicity.

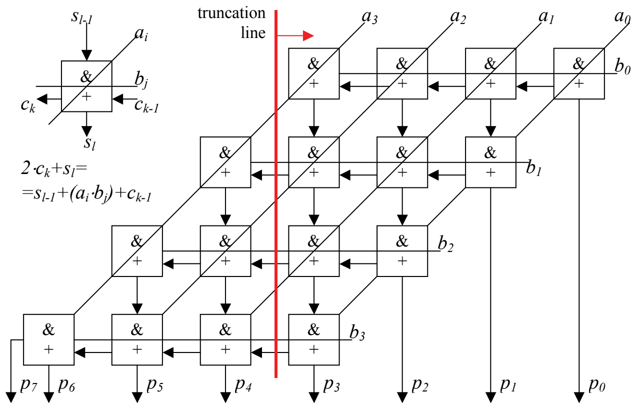

Fig. 1. Parallel array multiplier for the input width n=4

The most commonly used method to obtain a fixed-width multiplier is to implement full-width multiplier and then reduce its width by truncating n-LSBs. For example this method with additional rounding and normalization is employed for floating-point multipliers [5]. This method leads to the lowest error (only rounding error) however requires the highest hardware resources. Alternative solution is to truncate n-LSBs during multiplication process. In Fig. 1, the fixed-width multiplier is obtained by direct truncating all the logic less significant than 2^n, i.e. all the logic on the right side of the p_n is not implemented. This methods generates the highest calculation error, however significantly reduces the resources consumption. From Fig. 1, a conclusion can be drawn that more than a half of the hardware resources are truncated. This is conformed by [1] where in comparison to the full-width multiplier the following area reduction is obtained: 56% for n=6 and 51% for n=16.

Between full-width and direct-truncated multipliers, there are intermediate solutions, for which the truncation line is shifted w-bits right in comparison to the direct-truncated multiplier. Therefore, for the parameter w=0, direct-truncated multipliers is obtained, similarly for w=n, full-width multiplier is obtained. In should be noted that an increase of the parameter w results in lower calculation error at the expense of higher occupied hardware resources.

Alterative methods of reducing truncation error were presented in [1, 2]. These methods employ a special error compensation logic σ (see (34) [2]):

$$(1) \qquad \sigma = a_{n-2} \cdot b_1 + a_{n-3} \cdot b_2 + \ldots + a_1 \cdot b_{n-2} \ \ (+1)$$

These methods are based on error statistics. In the first approximation the error compensation value σ is equal the sum of the AND gates outputs. It should be noted that these

AND gates will be implemented in the multiplier if the parameter *w* is incremented by 1. Nevertheless, they will be taken with the weight equal ½ as the logic is shifted one bit to the LSBs. However in (1) these AND gates outputs are taken with weight equal 1, in order to compensate also for the less significant bits which are truncated.

For ASIC designs these additional AND gates and addition required to implement equation (1) occupy relatively insignificant resources – much less than for the parameter *w* incremented by 1. However when FPGAs implementation is taken into consideration, these AND gates require additional LUTs [3], which would increase hardware resources similarly like incrementing the parameter *w* by 1. Moreover, a multiplier implemented in FPGAs has different structure which causes that the equation (1) cannot be easily adopted in FPGAs.

FPGAs Implementation

The structure of the 2×4 multiplier implemented in FPGAs was presented in e.g. [6] and is given in Fig. 2 and Fig. 3. To obtain $n\times n$-bit multiplier, the $2\times n$ –bit multiplier should be replicated $\lceil n/2 \rceil$ -times and then the intermediate results added.

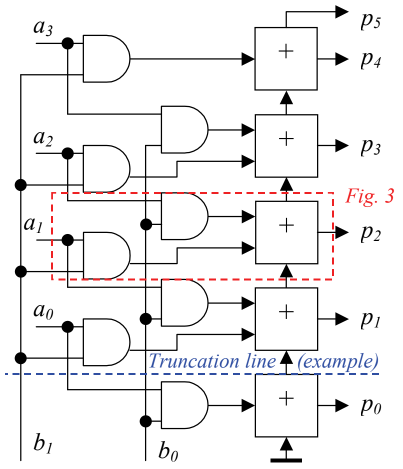

Fig. 2. Block diagram of the 2×4-bit multiplier

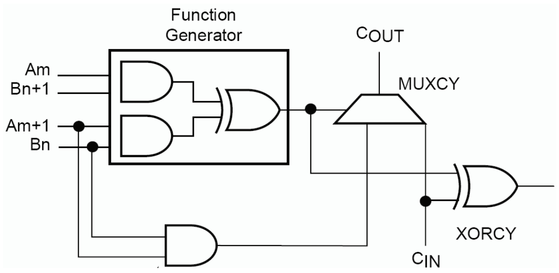

Fig. 3. Element from Fig. 2 implemented in Virtex family [6]

Similarly like for the multiplier presented for ASICs in Fig. 1, reduced-width multiplier can be implemented in FPGAs by truncating logic for less significant bits than p_{n-w}. Unfortunately, up to the authors knowledge no error compensation logic for FPGAs has been published until now.

Novel Error Compensation Structure

Assuming uniform probability of logic levels '0' and '1', a conclusion can be drawn form (1) that the expected value of the expression $c= a_{n-2}\cdot b_1 + a_{n-3}\cdot b_2$ (a fragment of the eq. 1) is equal ½. This expected value is the same as the expected value of any input bits a_{n-2}, b_1, a_{n-3} or b_2. Therefore the main idea behind this paper is to substitute the expression: $a_{n-2}\cdot b_1 + a_{n-3}\cdot b_2$ just by the either input: a_{n-2}, b_1, a_{n-3} *or* b_2. The FPGAs implementation of the proposed circuit is very similar to the direct truncated one, the only modification is to feed carry-in input with either of the input bits: a_{n-2}, b_1, a_{n-3} or b_2. Therefore the main benefit of the proposed structure is that it does not occupy any additional hardware resources in comparison to the direct truncation. The drawback of the proposed method is greater calculation error in comparison to [1, 2]. A block diagram of the proposed multiplier for *n*=8 and *w=1* is showed in Fig. 4, where four different $2\times k$-multipliers, similar to the presented in Fig. 2, are incorporated.

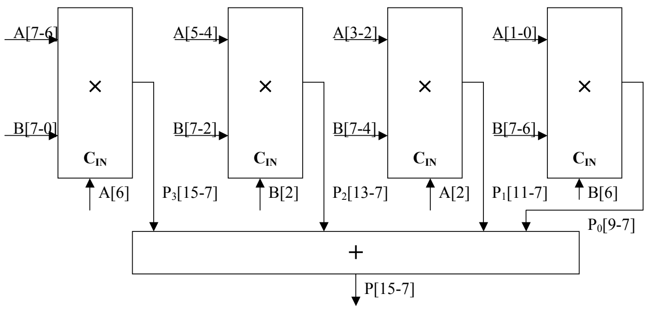

Fig. 4. Block diagram of the proposed multiplier for *w*=1, the four blocks of the multiplier are similar as in Fig. 2.

It should be noted that carry-in (C_{IN}) input to each $2\times k$-bit multiplier is fed interlacedly by input bits *A, B;* e.g. *A[i], B[j], A[i+4], B[j-4], A[i+8], B[j-8]…* This interlaced connection scheme results in a lower truncation error than in the case when only a single input is connected, e.g. *A[i], A[i+2], A[i+4]…* The next aspect is a selection of input *A* or *B* bits connected to the input C_{IN}. The authors' suggestion is to connect the most significant bit (MSB) which is partially truncated in the $2\times k$-bit multiplier, this input feeds only one AND gate (normally an input bit feeds two AND gates – see Fig. 2). For example in Fig. 4. and the leftmost $2\times k$-bit multiplier block, bit *A[6]* feeds only a single AND gate. Nevertheless, some more research might be conducted in order to find the best solution in each particular case; e.g. connecting *A[5]* instead of *A[6]* might results in lower average or maximum error for some parameter *w*.

Implementation Results

In order to properly evaluate the proposed multiplier, the following calculation error statistics are employed:

- ME (Mean Error): $ME = \frac{1}{N}\sum_{i=1}^{N} e_i$
- MAE (Mean Absolute Error): $MAE = \frac{1}{N}\sum_{i=1}^{N} |e_i|$
- RMSE (Root Mean Square Error): $RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_i^2}$
- Emax (maximal error): $Emax=MAX(|e_i|)$,

where e_i is the difference between the correct result (obtained by the full-width multiplier) and the obtained result for *i*-th sample. For *n*=8 all possible input combinations have been taken into account, therefore $N=2^{2\cdot n}= 2^{16}$.

Fig. 5 presents previously defined errors for multiplier input width *n*=8, different parameter values *w,* without and with the truncation error compensation logic. In order to distinguish whether the error compensation logic is applied, errors with error compensation logic have the suffix *c*, e.g. *emaxc* vs. *emax*. In order to make Fig. 5 readable, labels of different errors are sorted according to error values for *w=0.*

Fig. 5 presents results for unsigned numbers multipliers, therefore it cannot be directly compared with [1, 2]. The given error values are presented in Unit Last Place (ULP), i.e. for bit P_{n-w}. It should be noted that the ULP has the weight 2^{n-w} in comparison to full-width multiplier ($w=n$) and results present in [1, 2]. Therefore according to Fig. 5, *RMSE(w=0)*= 2.0 *ULP* and *RMSE(w=1)= 1.75 ULP*, thus RMSE(w=0)= 2.29 × RMSE(w=1). It should be noted that without the error compensation logic, Mean Error (ME) and Mean Absolute Error (MAE) are equal, thus the proper value is always greater or equal to the obtained value.

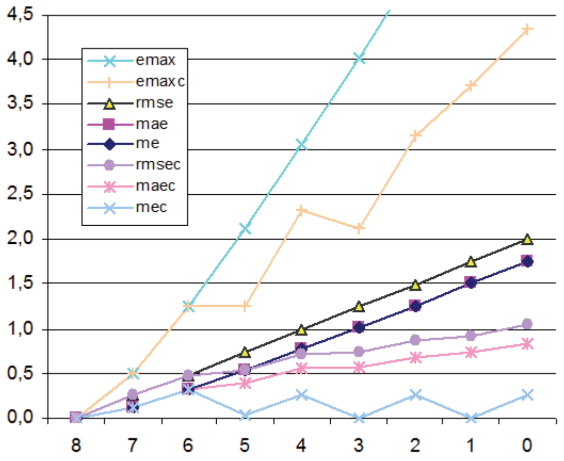

Fig. 5. Errors [ULP] for *n*=8 and different values of the parameter *w*

There is a significant difference of error values with and without the error compensation logic. For example for *w*= 0, maximum error *Emax* without the error compensation logic is equal 7.0 ULP (outside the range of Fig. 5) in comparison to *Emaxc*= 4.35 with the error compensation logic. The error reduction is especially noticeable for the mean error (ME), which e.g. for *w*=0 is equal: *MEC*= 0.25 and *ME*= 1.75. It should be noted that in some cases the mean error is crucial as the error value does not accumulate when ME=0. Furthermore, for *ME*=0, the intermediate errors compensate for the increasing number of summations.

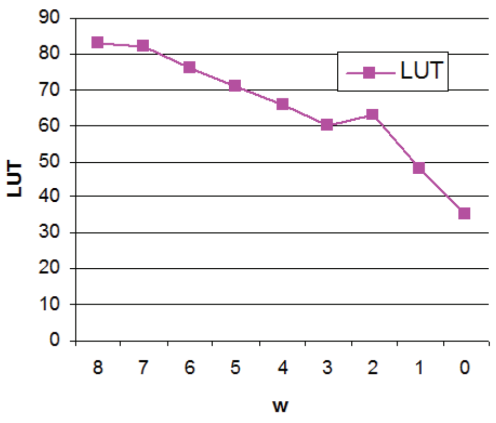

Fig. 6. FPGA resources (LUT) occupied by the multiplier for *n*=8 and different *w*

A conclusion can be drawn from Fig. 5, that the mean error with the error compensation logic, *MEC,* is roughly either equal 0 or 0.25 ULP. This holds as the proposed structure can compensate for only with a quantum 0.5, assuming uniform 0 and 1 logic levels probability. Therefore, whenever the condition *MEC*=0 is crucial and the obtained *MEC*≈0.25, then additional error compensation logic, A_i *AND* B_j, might be included. This logic mean value is equal ¼ for uniformly distributed input values. It should be noted that this logic requires additional FPGAs resources and introduces additional propagation time.

Fig. 6. presents occupied FPGA resources (number of Virtex-4 LUTs) for different parameter values *w*. It can be seen that the number of LUT modules decreases from 83 LUTs (w=8) to 35 LUTs (w=0).

Fig. 5 and Fig. 6 presents results for different values of the parameter *w*. Nevertheless a common practice is to employ fixed-width multiplier for which input and output width is equal *n*. In this case, the (*n*+*w*)-bit product is rounded to *n*-bits. Fig. 7 presents different kind of errors obtained in this case. The rounding error impacts dominantly the total error for *w*≥3. This holds as the rounding errors themselves for uniformly distributed inputs are: *ME*= 0, *MAE*= 0.25, *RMSE*= 1/(2·√3)≈ 0.289, *Emax*= 0.5. It should be noted that for *w*=0 no rounding logic is implemented therefore Fig. 7 and Fig. 5 are equivalent.

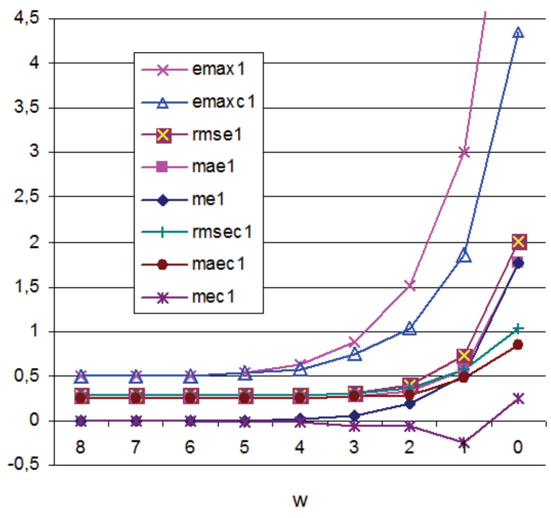

Fig. 7. Total error (including rounding error) for fixed-width multipliers (*n*=8)

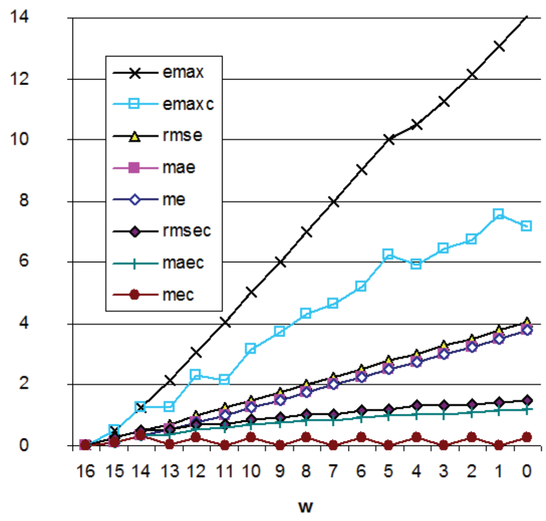

Fig. 8. Errors [ULP] for *n*=16 and different values of the parameter *w*

Fig. 8-10 presents similar results as in Fig. 5-7 but for *n*=16. Nevertheless for *n*=16 the number of all input values combination *N* equals 2^{32}, which would require too long simulation time. Therefore the number of tests *N* is limited to 100000 random numbers. The reduced number of tests might influence mostly the *emax* and *emaxc* values.

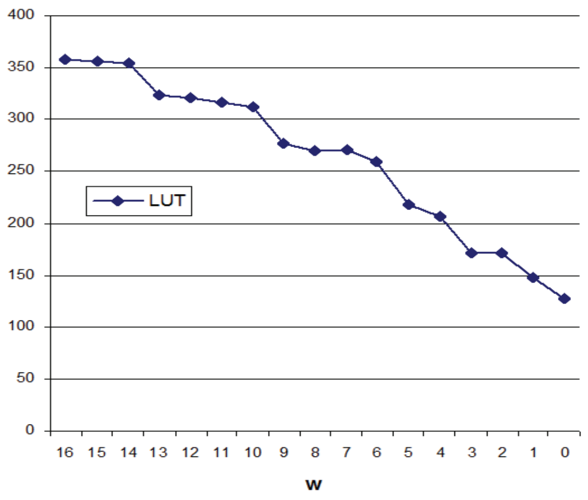

Fig. 9. FPGA resources (LUT) occupied by the multiplier for *n*=16 and different *w*

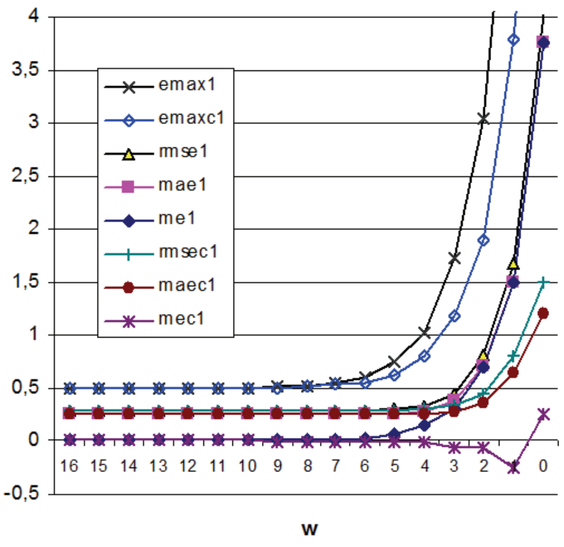

Fig. 10. Total error (including rounding error) for fixed-width multipliers (*n*=16)

Conclusions

This paper presents reduced-width multiplier with the novel error compensation method. This method does not require any additional FPGA resources, but significantly reduces truncation error. The next idea behind this paper is to promote reduced-width multipliers as they significantly reduce hardware resources. A designer may select proper parameter *w* as a compromise between greater computation error and occupied FPGA resources. Analyzing Fig. 7 and Fig. 10 for fixed-width multipliers, a conclusion can be draw that dominant part of the error is a rounding error for $w \geq 3$ and $n=8$ or $w \geq 4$ and $n=16$. Consequently, there is no reason to employ full-width multiplier unless compatibility with other systems is required. This is the case of floating point multipliers which must employ full-width multipliers in order to comply with IEEE-754 standard.

Publikacja finansowana ze środków na naukę MNiSW w roku 2009.

REFERENCES

[1] Lan-Da Van, Chih-Chyau Yang, *Generalized Low-Error Area-Efficient Fixed-Width Multipliers*, IEEE Transactions on Circuits and Systems, VOL. 52, NO. 8, pp. 1608-1619, August 2005

[2] Lan-Da Van, Shuenn-Shyang Wang, Wu-Shiung Feng, *Design of the Lower Error Fixed-Width Multiplier and Its Application,* IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 47, no. 10, pp. 1112-1118, OCTOBER 2000

[3] Xilinx, *Virtex-4 Family Overview*, www.xilinx.com, DS112 (v3.0) September 28, 2007

[4] J. Poldre, and K. Tammemae, *Reconfigurable multiplier for Virtex FPGA family* Int. Workshop on Field- Programmable Logic and Applications, Glasgow, Scotland, UK, pp. 359-364, Aug. 30 –Sept. 1, 1999.

[5] A. R. Omondi, *Computer Arithmetic Systems: Algorithms, Architecture and Implementation,* Prentice-Hall International, 1994

[6] S. Elzinga, J. Lin, V. Singhal, *Design Tips for HDL Implementation of Arithmetic Functions,* Xilinx Application Note XAPP215 (v1.0) June 28, 2000

__Autorzy__: dr inż. Ernest Jamro, Akademia Górniczo-Hutnicza, Kat. Elektroniki, Al. Mickiewicza 30, 30-059 Kraków; ACK Cyfronet, ul. Nawojki 11, 30-950 Kraków, E-mail: jamro@agh.edu.pl;
mgr. inż. Maciej Wielgosz, Akademia Górniczo-Hutnicza, Kat. Elektroniki, Al. Mickiewicza 30, 30-059 Kraków; ACK Cyfronet, ul. Nawojki 11, 30-950 Kraków, E-mail: wielgosz@agh.edu.pl;
prof. dr hab. inż. Kazimierz Wiatr, Akademia Górniczo-Hutnicza, Kat. Elektroniki, Al. Mickiewicza 30, 30-059 Kraków; ACK Cyfronet, ul. Nawojki 11, 30-950 Kraków, E-mail: wiatr@agh.edu.pl;