A DIVIDER-MULTIPLIER HIGH LEVEL SYNTHESIS LIBRARY ELEMENT FOR DSP APPLICATIONS

V. Rodellar[1], M. A. Sacristan[1], A. Alvarez[1], A. Díaz[2], V. Peinado[2] and P. Gómez[1]

[1] Departamento de Arquitectura y Tecnología de Sistemas Informáticos
Facultad de Informática- Universidad Politécnica de Madrid
Campus de Montegancedo s/n
Boadilla del Monte - (28660-Madrid) SPAIN

[2]Departamento de Arquitectura y Tecnología de Computadores
Escuela Universitaria de Informática- Universidad Politécnica de Madrid
Km.7 de la Carretera de Valencia
(28031-Madrid) SPAIN

ABSTRACT In this paper a divider-multiplier reusable library cell is presented. The division is implemented by means of an algorithm based on successive multiplications. The resulting structure uses a fixed-point format and two's complement arithmetic for operands of any size. It is coded with the VHDL synthetizable subset for the Synopsys™ Behavioral Compiler. The performance results in terms of area and time delay for different operand sizes and technologies are presented and discussed.

1. INTRODUCTION

A great challenge for circuit designers is Digital Signal Processing (DSP) area, due to the increment of algorithmic complexity of the problems, real time and low power restrictions in the final implementations. A tendency in prototyping design is to short the design cycle by means of techniques of high level synthesis and the use of cores previously tested, verified and validated as parts of a final complex problems [1]. General purpose and DSP processors normally implement complex operations, like division, by software. Interfering the real time achievement due to the difficulty of applying acceleration techniques [2]. A critical point in this class of functional units is the numerical data format representation. In general, floating-point arithmetic gives more accuracy than fixed-point arithmetic although more silicon area and less speed too. That's why DSP applications are implemented in fixed format when some tolerance in the accuracy is allowed. This paper develops and synthesizes an algorithm of division by means of successive multiplication using a previously designed fast multiplier/inner product unit. The resulting structure uses a fixed-point format and two's complement arithmetic. The same design can be used as a divider and multiplier able to operate with operands of any size. There are basically two methods for the implementation of the division based on multiplication: multiplication by the inverse and successive multiplications. The arithmetic unit developed here is based on he second method due to its better performance in speed compared with the first method. The basic idea consists in finding a set of factors F_i that multiplied by the divider makes the result equals to the unity. This method allows the use of re-coding techniques and numeric basis different to binary to increase the speed of the multiplication and as a consequence of the division operation. But the major inconvenient of the proposed structure is the impossibility of using segmentation techniques and then the results are not available in a clock cycle.

2. MULTIPLIER OVERVIEW

A multiplier-inner product generic library cell was previously designed and reused as core cell for the design of the divider [3], [4]. It allows any size for the operands X(m) and Y(n). It operates in fixed-point arithmetic and two's complement. The general structure of the multiplier can be seen in Fig.1. On it, three different stages can be identify, the generation and the addition of the partial products by means of redundant binary adders, the accumulation if the device is used as inner product unit and the conversion from binary redundant code to two's complement representation. The implementation can be done by three pipelined stages. This multiplier generates all the partial products at the same time and adds all of them by using a tree of redundant binary adders. Booth's algorithm is used in the generation of all the partial products in order to reduce them into a half. We have used redundant binary adders because they present the attractive characteristic of having an independent delay from the operands size and the area requirement is similar to a ripple-carry adder. This characteristic is essential in parameterized designs in order to avoid time performance degradation as the number of bit increases. This multiplier can be used also as inner product unit, then a binary redundant register is enclosed in the structure for this purpose. Finally, the result of m+n bits must be converted from redundant binary to two's complement and a fast carry select adder is provided in the structure to do this conversion. The performance of all system has been

0-7803-7057-0/01/$10.00 ©2001 IEEE.

also evaluated by using carry-look ahead and ripple-carry adders for this last stage [5]. The described functional unit when operates as multiplier overflow is not possible but when operates as inner product unit overflow may occur due to several multiplication results must be accumulated. In this last case, overflow detection is very complicated and implies a high cost in hardware.

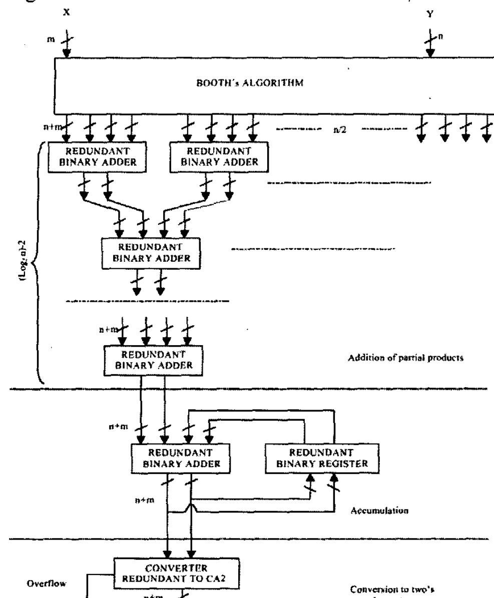

Figure 1. Multiplier Blocks diagram.

3. DIVISION ALGORITHM DESCRIPTION

A detailed description of this classic algorithm of division can be found in [6]. Next we will introduce the basic algorithmic aspects of the method. Given A as dividend and B as divisor, the algorithm consists in calculating a factor, denoted as F_i, in such a way that if we multiply A and B by F_i, we get after some iterations that the product of F_i by B is the unity. Then the resulting quotient is A by F_i:

$$\frac{A}{B}=\frac{A\times F_0\times F_1\times.\times F_i}{B\times F_0\times F_1\times.\times F_i}=\frac{A\times F_0\times F_1\times.\times F_i}{1}=A\times F_0\times F_1\times.\times F_i$$

The factor F_i is evaluated as follows:

$$F_i = 1+\left(\delta^{(2)^i}\right) \quad \text{with } \delta = 1-B \quad \text{for i = 0}$$

By observing the initial value of δ (i=0), we can deduct that the value of B must be quoted. So, the algorithm must start with the normalization between 1 and 0,5 of B. Values of B upper than 1 make the algorithm diverging and values of B lower than 0,5 reduce the convergence speed of the algorithm. Multiplying or dividing by 2, dividend and divisor, can do this normalization, implemented by right-left shift register. But this simple solution is very time consuming, a way of completing the normalization in a clock cycle would be to use a barrel shifter, but this solution is very area consuming. Then we have adopted an intermediate solution, consisting in a left–right barrel shifter able to shift four positions in a clock cycle. The number of clock cycles to complete the number normalization will be:

NCB = max {integer (FDN/4), (integer (IDN/4) + 1)}

FDN is the number of digits of the fractional part and IDN is the number of digits of the integer part of B. In order to state the control of the resulting circuit the end of the algorithm must be detected. This can be done in two ways. The first one consists in checking the condition $B\ F_i$ = 1.0 and the second one is to determine the loop order i in the worst case. The second possibility depends on the accuracy of the binary representation being used, which is the adequate in our case, because the multiplication result must be truncated to the chosen data format. And the truncation can produce that B never reaches the unity. For instance, if we use a 24-bit fixed-point two's complement arithmetic representation with sixteen bits for the decimal part and eight bits for the integer part, it gives a range of representation from –128 to 127,9999847412, being 0.0000152588 the smallest number to be represented with this format. Taking B = 0.5, which represents the worst case, following the evolution of δ_i, can be observed that for i = 4, δ_i reaches the smallest possible value able to be represented with the chosen data format. If we continue looping, the rest values of δ_i will be rounded to zero, so the maximum number of loops that the algorithm must iterate can be established as 5. In a generic case the number of iterations can be evaluated as:

$$\text{IN} = \text{integer}(\log_2(\text{FDN})) + 1$$

In the case that B would be different than the unity and $\delta_i = 0$, a problem may occurs when using this loop end criteria. This problem can be solved by checking the value of $\delta_i = 0$, which can be easily done by means a wired OR of all its digits. Of course a little error in the less significant bit must be assumed in the result of the division.

A detailed flow chart of the division algorithm is shown in Figure 2. First of all, we check if the dividend and/or the divisor are zero in order to finish the operation, this can be implemented by a means of a wired OR. If the dividend is an equal to zero the result is zero. If the divisor is an equal to zero the result is the maximum positive number or the minimum negative number according to data format and depending on the dividend sign, in this case the overflow signal is activated. The next step is to see if B is quoted in the adequate interval and to start the normalization if necessary. If normalization is necessary overflow and underflow occurrences in A

must be tested. The initial values of the pivoting elements, δ_i and F_0 are calculated and then the algorithm division loop starts. First, the next value of δ_i is calculated and checked the condition $\delta_i = 0$. If this is true, the operation ends and the result of the division is $A_{i-1} F_{i-1}$. But if it is false, the next values of B_i, A_i and F_i are calculated. The algorithm ends when B = 1.0 in other case returns to calculate the next value of δ_i. The order of calculating A_i and B_i has been reversed due to the pipeline structure of the multiplier.

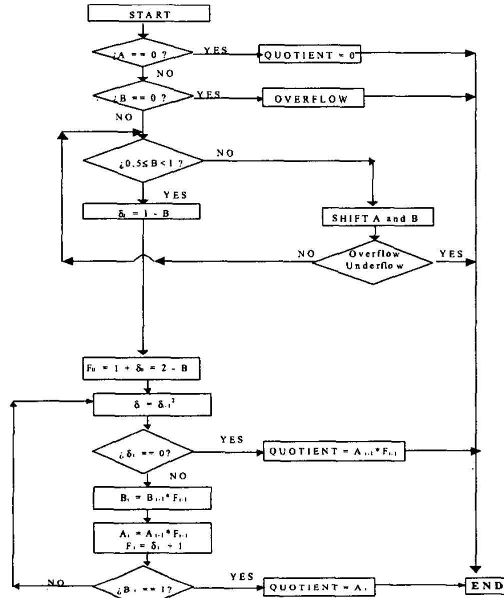

Figure 2. Detailed division algorithm flowchart

4. DIVISOR STRUCTURE

The divider blocks diagram is shown in Fig. 3. The basic functional units are registers, Mux, adder/subtractor and the multiplier already described. The registers that store the initial values of the operands A and B have the characteristic of behave as a 4 positions barrel shifter. The pipelined structure of the multiplier condition the order of the multiplications: $\delta_{i+1}=\delta_i^2$, B.F$_i$, B.F$_i$. As the multiplier has three stages of pipeline, the maximum number of clock cycles to complete the kernel calculations of an operation of division will be:

$$MCC = 1 + (3\ IN)$$

Three clock cycles are devoted to the realization of the proper multiplication and one cycle to overflow correction after sign bits truncation. Next, we will analyze if possible situations of overflow may occur. First, let's see the range of values of the implicated parameters. B and δ_i will be always less than the unity because B is normalized in the range of [0.5,1), δ=1-B and A can have any value.

The range for F will be [1.5, 1) due to $F = 1 + \delta$. Next, we will see the implicated operations in the basic functional units of the divider with the values of these parameters:

- *Normalization:* An overflow is produced when the dividend (A) is very big and the divisor (B) is very small. The result is the biggest number able to represent with the chosen data format. The result sign will depend on the operand signs. An underflow may also occur, it must be activated when A has a value different than zero before normalization and its value changes to zero after a shifting operation in the process of normalization. In this case, the result is zero.

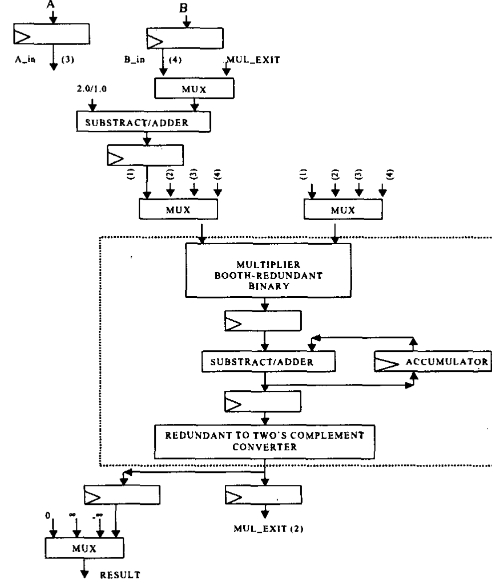

Figure 3. Divider Blocks Diagram

- *Adder/Substractor:* This functional unit will calculate the operations: 1 - B, 2 - B, and 1 + δ. Overflow never happens according to the values of B and δ mentioned above.
- *Multiplier:* The basic calculations done by the multiplier are: $A.F_i$, $B.F_i$, and $\delta.\delta$, and they never can produce an overflow. But $A.F_i$ must be checked after the truncation of the integer bits part to verify if all truncated bits correspond to bits sign. In other case, the algorithm must finish and the result will be the maximum or minimum value according to the sign of A and B.

The maximum number of clock cycles demanded in the main stages of the algorithm is:

- Start: 1 cycle
- Normalization: NCB cycles
- Multiplication: (1 + 3 IN) cycles
- End: 1 cycle

So, the worst case to complete an operation of division can be evaluated as follows:

WD = 3 + NCB + 3 IN clock cycles

The duration of each clock cycle obviously depends on the technology used for its implementation.

6. IMPLEMENTATION RESULTS

In order to compare the performance in terms of area needs and critical path delay, the implementation of the circuit was done under two different technological points of view. The first one was to mapping the resulting net list after synthesis on a FPGA. And the second one was to mapping the same word size designs on a 0,8 μ library from AMS. The chosen FPGA was from VIRTEX family device XCV150-4-pq240. Its main characteristics are, maximum clock frequency in optimal conditions 200 MHz, number of slices available 1728, total number of logical gates 164 674, this amount includes also the input/output logic. The results for this implementation are shown in Table 1.

Size	Decimal Digits.	Slices	Slices use (%)	Number of gates	Delay (ns)
8	4	358	20	5202	27.710
8	6	350	20	5070	26.800
16	4	940	54	12788	37.471
16	8	931	53	12600	38.601
16	12	915	52	12430	38.229
24	4	1716	99	21696	49.922
24	8	1706	98	21502	43.250
24	12	1697	98	21392	45.008
24	16	1680	97	21168	47.006
24	20	1672	96	21058	44.795

Table 1. Implementation results for FPGA

Size	Decimal Digits	Area in μ^2	Delay (ns)
8	4	6769.005371	11.24
8	6	6641.704590	11.24
16	4	17492.00000	14.61
16	8	17094.421875	14.67
16	12	16884.964844	14.67
24	4	42052.398438	17.48
24	8	41716.984375	17.73
24	12	41548.429688	17.09
24	16	40931.632812	17.09
24	20	40770.687500	17.15

Table 2. Implementation results for 0,8μ technology

On it can be noticed that as the number of decimal digits increases the number of slices and the number of logical gated needed for the implementation decreases. The reason for this decrement is due to the fact that the number of bits in the integer part decreases, then the circuit responsible for the sign bits truncation is more simple and as a consequence needs less hardware. But this same reduction is not observed in the critical path delay. For instance by comparing the 24 format data format with 12 and 16 bits of fractional part, we obtain a delay of 45.008 ns and 47.006 ns. respectively. This implicates that the reduced pieces of hardware were not located in the critical path delay. The mean operating frequency for a 24-bit circuit is around 22 MHz, which represents the 12% of the theoretical maximum speed given in the FPGA specifications.

The results for 0,8μ implementation are shown in Table 2. The area and critical path delay results follow the same tendency than in the case than the FPGA implementation. And the mean operating frequency for the case of 24 bits is 57 MHz.

7. CONCLUSIONS

The principal result of this work is the design of a divider-multiplier under the scope of its reusability. In order to palliate and excessive performance degradation with the value increment of the parameterized characteristics a special care must be taken in the algorithmic implementation details of the circuit. Some improvements could be done in order to achieve better performance results. First, all synthesis runs where done with low level effort, then the designs are not optimized. And in the case of FPGA automatic place and route with low level effort was also used. Next, a barrel shifter wider that 4 bits could improve time delay, but its optimal dimension depends on the normalization process. Then its size could be parameterized according to data range for each specific application. As the adders/subtractors implemented are carry select type and they are very fast, the use of faster adders is under study. And finally the use of two multipliers could allow some degree of parallelization in the algorithm with the consequent improvement in speed but degradation in area.

8. ACKNOLEDGEMENT

This work is being funded by grants CAM 07T/0001/2000 and TIC99-0960 from the Spanish Comisión Interministerial de Ciencia y Tecnología.

9. REFERENCES

[1] M. S. Ben Romdhane, V. K. Madisetti and J. W. Hines, *Quick-Tournaround ASIC Design in VHDL*, Kluwer Academic Publishers, 1996.

[2] R. Omondy, *Computer Arithmetic System: Algorithms, Architectures and Implementations*, Prentice Hall, 1994.

[3] M. A. Sacristan, V. Rodellar, A. Diaz and P. Gómez, *A Reusable Inner Product Unit for DSP Applications*, Proceedings of the 25th Euromicro Conference, Volume I, pp. 209 -213, 1999.

[4] X. Hwang, W. J. Lui and B. W. Wei, *High-performance VLSI Multiplier with a New Redundant Binary Coding*, J. VLSI Signal Processing, Vol.3, 1991, pp.283-291.

[5] V. Rodellar, M. A. Sacristan, A. Diaz, V. Peinado and P. Gómez, *Performance Evaluation of Reusable Multipliers for Rapid Prototyping*. 43th Midwest Symposium on Circuits and Systems. Michigan State University. Lansing, Michigan (USA), August 8-11, 2000. (Proceedings pending of publication).

[6] K. Koren, *Computer Arithmetic Algorithms*, Prentice Hall, 1991.