Revista EIA
ISSN: 1794-1237
revista@eia.edu.co
Escuela de Ingeniería de Antioquia
Colombia

División Decimal Parametrizable usando Lenguaje de Descripción de Hardware

López, Jorge Hernán; Restrepo Cardenas, Johans; Tobon Gomez, Jorge Enrique
División Decimal Parametrizable usando Lenguaje de Descripción de Hardware
Revista EIA, vol. 17, núm. 33, 2020
Escuela de Ingeniería de Antioquia, Colombia
Disponible en: https://www.redalyc.org/articulo.oa?id=149263063017
DOI: https://doi.org/10.24050/reia.v17i33.1318

División Decimal Parametrizable usando Lenguaje de Descripción de Hardware

Parametric Decimal Division using Hardware Description Language

Divisão decimal paramétrica usando a linguagem de descrição de hardware

Jorge Hernán López jhernan.lopez@udea.edu.co
Universidad de Antioquia, Colombia
Johans Restrepo Cardenas johans.restrepo@udea.edu.co
Universidad de Antioquia, Colombia
Jorge Enrique Tobon Gomez jorge.tobon1@udea.edu.co
Universidad de Antioquia, Colombia

Revista EIA, vol. 17, núm. 33, 2020

Escuela de Ingeniería de Antioquia, Colombia

Recepción: 12 Mayo 2019
Aprobación: 15 Enero 2020

DOI: https://doi.org/10.24050/reia.v17i33.1318

Redalyc: https://www.redalyc.org/articulo.oa?id=149263063017

Resumen: En este trabajo se describe un algoritmo rápido y de alta precisión escrito en el lenguaje de descripción de hardware, VHDL para realizar la división entre dos números decimales, es decir, los números compuestos por una parte entera y una decimal, bajo el esquema de una representación de punto fijo. El algoritmo propuesto no es una aproximación, como se hace en la mayoría de los casos, escogiendo el algoritmo según la necesidad propia, en tiempo o en área de lógica. Para ello, el tamaño de los bits de los operandos se puede ajustar mediante un par de parámetros N y M, según los cuales dependerá la latencia del cálculo. El proyecto se sintetiza finalmente en una matriz de puertas programables o FPGA del tipo SPARTAN 3E de XILINX.

Palabras clave: VHDL, FPGA, OPERACIÓN, DIVISIÓN.

Abstract: In this work we describe a fast and high-precision algorithm written in VHDL Hardware Description Language to perform the division between two_nite decimal numbers, i.e. numbers composed of an integer part and a decimal one, under the scheme of a fixed point representation. The algorithm proposed is not an approximation one as it is usually considered. To do so, the size of the bits of the operands can be tunned by means of a couple of parameters N and M, according to which the latency of the calculation will depend. The project is _nally sinthesized in a _eld programmable gate array or FPGA of the type SPARTAN 3E from XILINX.

Keywords: VHDL, FPGA, OPERATION, DIVISION.

Resumo: Neste trabalho descrevemos um algoritmo rápido e de alta precisão escrito na linguagem de descrição de hardware, VHDL, para realizar a divisão entre dois números decimais, ou seja, os números compostos por uma parte inteira e uma parte decimal, sob o esquema de um representação de ponto fixo. O algoritmo proposto não é uma aproximação, como é feito na maioria dos casos, escolhendo o algoritmo de acordo com a própria necessidade, no tempo ou na área lógica. Para isso, o tamanho dos bits do operando pode ser ajustado por um par de parâmetros N e M, dependendo de qual dependerá a latência do cálculo. O projeto é finalmente sintetizado em uma matriz de portas programáveis ou FPGA do tipo SPARTAN 3E da XILINX.

Palavras-chave: VHDL, FPGA, OPERAÇÃO, DIVISÃO.

1 Introduction

From the four basic arithmetic operations, namely addition, substraction, multiplication and division, the only one not implemented up to now in

FPGA arrays as a built-in or primitive function is division. In that sense, it is not considered as a primordial operation despite of being essential part of more elaborated functions like averages, statistical analyses, digital processing of signals and images, algorithms or simulations, etc. This is the reason why the division as algorithm is based on multiplication, an operation with a higher hierarchy, which in turn involves approximation methods. More concretely, division appears in programing languages as a high expensive operation and very time consuming. In FPGA devices, this operation has been addressed in different ways by using for instance repeated operations of substraction [1], Taylor series [2], iteration of multiplications [3], or by means of some algorithms like the Goldschmidt algorithm [4], the CORDIC one [5], or the Vedic method [7], etc. Every single attempt is based on approximation methods involving their own errors, which in turn, in the process of minimizing them, more hardware surface or more iterations are needed resulting in a greater computational cost. Moreover, such algorithms are designed for a particular numerical representation and a change in the representation of an output implies a redesign of the corresponding HDL module.

According to this, a new arithmetic module to perform divisions under the

scheme of a fixed point representation is proposed.

2 FIXED POINT REPRESENTATION

In the fixed point representation of a decimal number, two integers parameters M and N are used. The former stands for the size or number of bits used for representing the integer part whereas of parameter N is that corresponding for representing the decimal one. As long as our problem concerns to the division between two decimal numbers, the result or quotient is another decimal number where the residue is not included in the operation.

For a glance, we can consider the problem of writing the number 17,345 with a word of 24 bits where 8 bits are dedicated for representing the integer part, i.e. M = 8 and therefore N = 16 bits are reserved for the decimal part. Thus, the binary representations of the integer and decimal part are respectively $17_{10} = 00010001_2$ and $0{,}345_{10} = 0{,}0101100001010001_2$. According to the parameterization chosen (M=8 y N=16), the complete number can be written as follows: $17{,}345_{10} = 00010001{,}0101100001010001_2$. At this point we have to stress that the binary representation of the example, does not correspond exactly to $17{,}345$ but to $17{,}3449859619140625$. On this regard, the difference is however smaller than the less significative bit of the decimal part, i.e.:

$$|17{,}345 - 17{,}3449859619140625| \leq 2^{-16} \quad (1)$$

$$0{,}0000140380859375 \leq 0{,}0000152587890625 \quad (2)$$

In order to optimize the number of bits can be allocated for the different operands in the division, a maximum size of 2M + N bits either for dividend or quotient was reserved whereas for the divisor a size of M + N bits was considered.

Therefore, the maximum decimal value can take either the dividend or the quotient is:

$$\left(\sum_{m=0}^{2M-1} 2^m\right),\left(\sum_{n=1}^{N} 2^{-n}\right) \qquad (3)$$

[3]

By adding the integer and decimal parts, it is easy to show that such a number corresponds to:

$$\frac{2^{2M+N}-1}{2^N} \qquad (4)$$

[4]

Analogously, for the divisor, we have:

$$\left(\sum_{m=0}^{M-1} 2^m\right),\left(\sum_{n=1}^{N} 2^{-n}\right) \qquad (5)$$

[5]

which corresponds to the following number:

$$\frac{2^{M+N}-1}{2^N} \qquad (6)$$

[6]

In any case, the maximum resolution of the operation depends only on the amount of bits needed for the decimal part as follows:

$$\delta E = 2^{-N} \qquad (7)$$

[7]

It must be stressed that such a resolution does not depend on the hardware surface neither on the execution time as it occurs with other methods.

Parameters M and N are programmed from the GENERIC platform of the designed entity and their initial values must be entered before implementing the module. Such parameters allow to the programmer to make a design of the module according to the needs. Here, it must be stressed that in practice, every single calculation has particular conditions

about the numbers participating in the operation, and hence every particular process has its own range and resolution.

3 ALGORITHM

The algorithm mimics the way as division is taught in primary schools. To visualize the method in a clear way, we can consider first the division between two integer numbers (decimal base) B ÷ A where B is the dividend and A is the divisor. Division implies:

$$B = A * Q + R \qquad (8)$$ [8]

Where Q is the quotient and R is the remainder, both of them integers. The algorithm for division proceeds as follows:

1. By starting from the most significative digit in the dividend, a number of digits equals to those contained in the divisor is taken. If the corresponding number is smaller than the divisor, an additional digit in the dividend is considered.

2. The integer corresponding to the number of times the divisor is contained within the dividend is computed. Such a number is then multiplicated by the divisor and the result is then substracted from the one taken in the dividend in the previous step. The difference is stored from right to left, and at the end, such a number resulting from concatenation corresponds to the quotient.

3. To the result of every substraction in the previous step, the most significative digit of the dividend is added to the less significative digit of the substraction and the iteration proceeds to step 2 until the divisor contains no more digits.

Analogously, the steps to compute a division between two binary numbers B ÷ A are:

1. Identification of the size of divisor. To do that, zeros located at the left of the binary number are deleted, i.e. the amount of bits having the divisor after the first most significative bit equals to '1' is counted and the result is carried to a register W.

2. The most significative part of the dividend, with a size equals to that of the previous register W, is then taken to another register S.

3. The substraction of the two registers is carried out:

$$\boldsymbol{C = S - W} \qquad (9)$$ [9]

4. The register C is evaluated in such a way that if $c \geq 0$ then the result corresponding to C is assigned to S and concomitantly a logic '1' is concatenated to the right to a new register Q. Otherwise, if $c \leq 0$, a logic '0' is concatenated to the right to Q.

5. The next most significative value of the register of dividend is concatenated to the right of register S and the process returns to step 3.

Iteration finishes when the register of the dividend does not contain any more bits.

In the case the divisor is equal to zero, an error signal is activated indicating the operation can not be performed and the process ends up. The algorithm presented is designed for positive numbers, however the module can be employed to work with negative numbers by adding a process which makes the conversion to positive numbers, and at the end, the correct sign is given according to the conventional rule of product of signs.

4 FPGA IMPLEMENTATION

The division algorithm was implemented using an FPGA Spartan 3E-500 of XILINX [8]. The hardware surface used, concretaly the amount of flip-flops and Look-Up Tables (LUT), depends on the selected values for the parameters M and N.

Figure 1 reveals a closely linear dependence of the hardware surface resources as the integer part M of the divisor increases by keeping constant N.

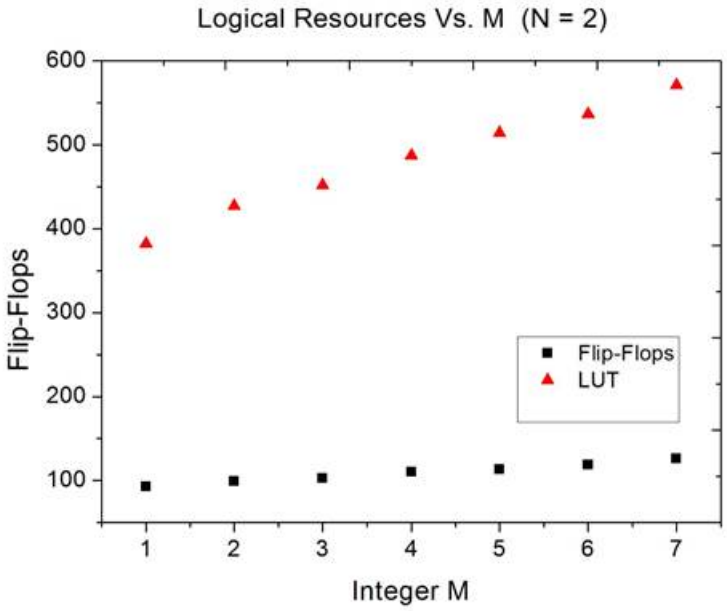

Fig. 1
Logical resources measured in Flip-Flops units as a function of the parameter M at a constant value N =2.

Analogously, the parameter M was kept constant whereas the parameter N was varied. the resultant behavior, which is also of a linear type, is shown in figure 2.

Figures above show how the use of bigger numbers entails a greater logical surface in the FPGA when no tool is used to decrease such area. On the other hand, the latency of the calculation is 3(M+N) clock cycles, which means the maximum frequency the algorithm can operate can

range between 75 MHz and 80 MHz depending on M and N values. Such a frequency can be even higher by using the tools provided by the executing platform of the HDL language, which in this case corresponds to ISE-XILINX.

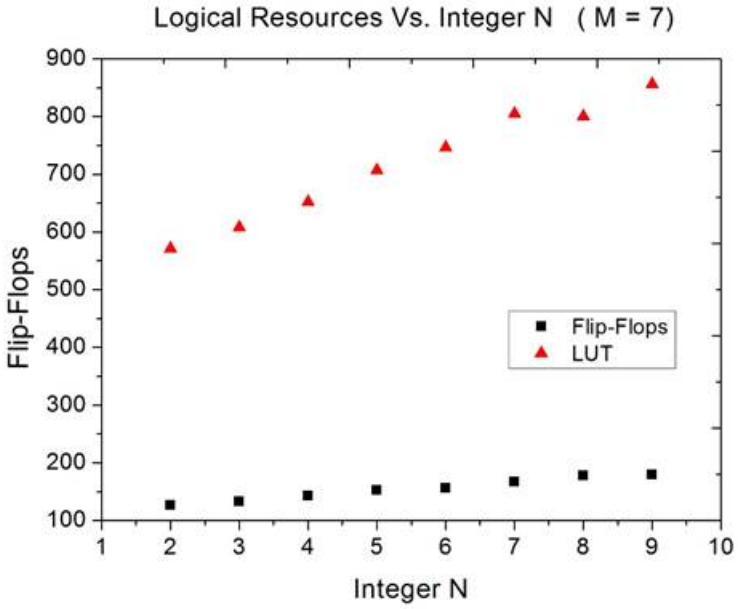

Fig. 2
Logical resources measured in Flip-Flops units as a function of the parameter N at a constant value M=7.

The corresponding flowchart implementation for division in the FPGA can be observed in figure 3. In the state S0, when the variable Enable is activated, the registers Dividen_int and Divisor_int are loaded with the entries DIVIDEND and DIVISOR respectively. In state S1 the size of divisor is calculated whereas in the following state S2, the register S is loaded with the most significative part of Dividen_int and of the same size as Divisor_int. In state S3the value Divisor_int is substracted from S and the result is stored in register R. In state S4 the value of R is analyzed in such a way that if $R \geq 0$, concatenation of a logical '1' to the right of register Q is performed. Otherwise, if $R < 0$, concatenation is carried out with a logical '0'. Such a concatenation takes place in state S5.

Finally, in state S6, concatenation to the right of register S is made with he following most significative value of Dividen_int. The process stops when no more digits of Dividen_int are available, calculation ends up and the result is recorded in register Q. This las step corresponds to state S7.

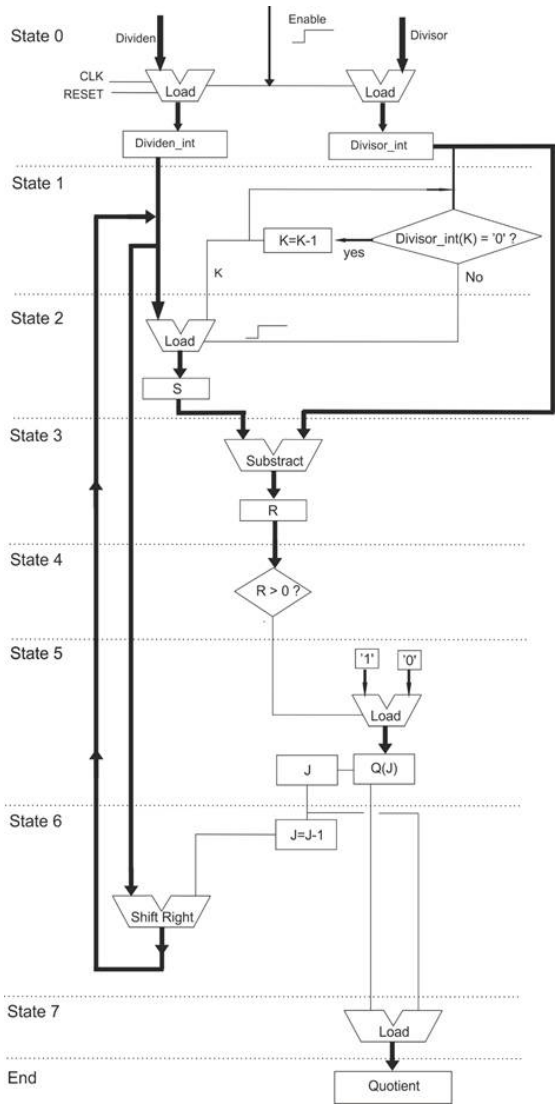

Fig. 3
Flowchart for the division algorithm implemented in an FPGA gate array.

CONCLUSIONS

The parameterization property of the algorithm allows the designer to instantiate a module that suits his own needs. Knowing the maximum values that the numbers to be divided can take, the value of the parameter M (integer part of the number) can be chosen, while the value of N (decimal part of the number) is taken by taking into account the desired resolution in the result.

The module presented also works to perform the operation between integers, in that case, an output is added to the module, Residue, containing the last recorded value of register S (Figure 3). If the division is exact the value of the residue is zero. Configured in this way, the module can be used to calculate the module operation.

The designed algorithm allows a higher precision computation than the modules that are based on approximation algorithms. In our algorithm

the error depends on the resolution chosen, while in others, the error depends on the number of times the iteration is done or the number of operations, in any case, in order to achieve the same resolution, more hardware surface of logical elements or greater processing time are required.

Implementation in the FPGA was done without the use of path or área minimization tools, so the program can be optimized even more by reducing either the work area or the physical paths of the signals, or by increasing the frequency at which the FPGA clock can operate.

The algorithm can be pipelined, which will increase logical needs, but it reduces the operation time even to a clock cycle, which is useful when the processing time is critical.

Finally, we used the Simulink software from Matlab and Modelsim. Different divisions were simulated at random and the results obtained were compared with the results of the operation, always keeping the error less than or equal to the resolution chosen.

Agradecimientos

Support provided by the CODI-UdeA project 2016-10085 and the exclusive dedication UdeA program to one of the authors (J. R) is acknowledged.

References

1. A. H. Karp, P. Markstein.,1997. High Precision Division and Square Root, ACM Transactions on Mathematical Software (TOMS), Vol.23(4), pp.561589, 1997. DOI : 10.1145/279232.279237
2. T. J. Kwon, J. Draper., 2008. Floating-Point Division and Square Root Implementation Using a Taylor-Series Expansion Algorithm With Reduced Look-Up Tables, Proc. 51st Midwest Symp. Circuits Syst., pp. 954957, 2008. DOI: 10.1109/MWSCAS.2008.4616959
3. H. Nikmehr, B. Phillips, and C. C. Lim., 2008. A novel Implementation of Radix-4 Floating-Point Division Square-Root Using Comparison Multiples, Computers and Electrical Engineering, vol. 36(5), pp. 850863, 2010. DOI: 10.1016/j.compeleceng.2008.04.013
4. R. Goldberg, G. Even, and P. M. Seidel., 2007. An FPGA Implementation of Pipelined Multiplicative Division With IEEE Rounding, 15th Annual IEEE Symposium on Field Programmable Custom Computing Machines FCCM, pp. 185196, 2007. DOI: 10.1109/FCCM.2007.59
5. S. Pongyupinpanich, F.A. Samman, M. Glesner and S. Singhaniyom., 2012. Design and Evaluation of a Floating-Point Division Operator Based on CORDIC Algorithm, Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON), 9th International Conference on, pp. 1618, 2012. DOI: 10.1109/ECTICon.2012.6254331
6. A. J. Thakkar, A. Ejnioui., 2006. Pipelining of Double Precision Floating Point Division and Square Root Operations, Proceedings of the 44th Annual

Southeast Regional Conference On ACM-SE 44, Melbourne, Florida, 2006. DOI: 10.1145/1185448.1185555

7. D. Rutwik., 2004. V.S. Kanchana. Low Power Divider Using Vedic Mathematics. IEEE, Advances in Computing, Communications and Informatics. 2014 International Conference on, 2004. DOI: 10.1109/ICACCI.2014.6968436
8. F. Adamec, T. Fryza., 2009. Binary Division Algorithm and Implementation in VHDL, Proceedings of 19th International Conference Radioelektronika 2009, pp. 8790, 2009. DOI: 10.1109/RADIOELEK.2009.5158757
9. J. Liu, M. Chang and C. Cheng.,2006. An Iterative Division Algorithm for FPGAs, Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays California, USA, 2006. DOI:10.1145/1117201.1117213
10. M.D. Ercegovac and R. McIlhenny., 2008. Design and FPGA Implementation of Radix-10 Algorithm for Division with Limited Precision Primitives. Proc. 42nd Asilomar Conference on Signals, Systems and Computers, 2008. DOI: 10.1109/ACSSC.2008.5074511
11. S. F. Oberman and M. J. Flynn., 1997. Division Algorithms and Implementation, IEEE Trans. On Comp, vol. 46, pp. 833854, 1997.
13. M. Franke, A. T. Schwarzbacher and M. Brutscheck., 2007. Implementation of Different Square Root Algorithms, Proc. 6th IEEE Electron. Circuits Syst. Conf., pp. 103106, 2007.