Implementation of high-speed fixed-point dividers on FPGA

Nikolay Sorokin
nus@mail.khstu.ru
Pacific National University,
Tikhookeanskaya str., 136, Khabarovsk, 680035, Russia

ABSTRACT

Study deals with implementations of fixed-point division modules based on different algorithms on basis of Xilinx FPGAs. We show that our implementation of the non-restoring algorithm is significantly faster and smaller than the 32-bit IP Core "Pipelined Divider" from Xilinx. For example, the speed of the 32-bit designed module is almost 245 MHz vs. 193 MHz from Xilinx divider. Moreover, high-speed parameterized modules are designed to provide arbitrary precision of the fixed-point division, for example, with 64-bit or 128-bit operands and large fixed-point result.

Keywords: high-precision computations, fixed-point division, modular design, programmable logic, FPGA

1. INTRODUCTION

Nowadays Field Programmable Gate Arrays (FPGAs) are frequently used for complex System-on-Chip (SoC) designs. They are oriented for the automotive control, online data processing and a wide range of computational tasks. Division operation is used in these tasks very often, e.g., for computing the coordinates of an object or a point on a grid in real-time scale. Because the result of the division operation in many cases is an approximate value, this can influence the solution and lead to the subsequent fault results.

Fixed-point arithmetic is often used for the computations because of the cost reasons – floating point calculations significantly increase the size of the design and make it more complex. But the existing solutions for the fixed-point division are bounded, e.g. the input and output widths of the modules are limited. Among the solutions in this field the Xilinx IP Core Divider is presented. This core has 32-bit input operands and can produce at most 32-bit fractional remainder. These widths of the operands are not practical in many applications, which require higher precision during the calculations. Another problem of the standard Xilinx solution is that the divider module occupies a big area on the chip and is not a high-speed design.

This paper focuses on implementation of high-speed fixed-point division modules for FPGA. Implementations of different algorithms on Xilinx FPGA are investigated. Results for different precision of the implemented modules are analyzed and compared with the standard division core from Xilinx.

This paper is organized as follows. Section 2 gives the introduction into some standard division algorithms and optimization techniques. Section 3 describes the implementation of the algorithms on Xilinx FPGA base. The results and discussion are presented in Sections 3 and 4.

2. FIXED-POINT DIVISION METHODS

Division algorithms

The task of the division is to compute the quotient $Q \in Z_{\geq 0}$ and the remainder $S \in Z_{\geq 0}$ after division $Y \in Z_{\geq 0}$ by $D \in Z_{>0}$ such that $Q=int(Y/D)$ and $S = Y - Q \cdot D$ under the condition $S < D$. The division is a series of subtractions of the divisor from the dividend and from the partial remainder values.

As it is widely known, the standard fixed-point algorithm follows a "paper-and-pencil" technique: every iteration it produces the fixed number of quotient bits. This involves the addition, multiplication and shift operations. Normally, the dividend is a *2n*-bit integer value (e.g., denoted by $Y(y_{2n-1}\, y_{2n-2} \cdots y_1\, y_0)$) and divisor, quotient and remainder are *n*-bit values with $n \in Z_{>0}$ accordingly. In this study, positive (unsigned) values of the operands are assumed, which is not a considerable restriction for the results of this work.

The most popular division algorithm is called a restoring algorithm described in, e.g., [1,6]. After each iteration the algorithm subtracts the divisor from the partial remainder, and if the result is less than 0, the previous value must be restored. This leads to the restoring of the partial remainder on the different steps and is a drawback of this algorithm.

The division operation in the restoring algorithm is done in *n* iterations and formally is written as follows:

$$S^{(i)} = 2S^{(i-1)} - q_{n-i}(2^n D), \quad (1)$$

with $S^{(0)} = Y$, $S^{(n)} = 2^n S$

and $q_{n-i} = \begin{cases} 1, \text{if } (2S^{(i-1)} - 2^n D) \geq 0 \\ 0, \text{if } (2S^{(i-1)} - 2^n D) < 0 \end{cases}$.

After *n* iterations we have

$$S^{(n)} = 2^n S^{(0)} - Q(2^n D) = 2^n [Y - (Q \times D)] = 2^n S. \quad (2)$$

The quotient after the division of *2n*-bit number by an *n*-bit number can have more than *n*-bits. In this case the overflow occurs and the condition $Y < 2^n D$ must be checked before the division.

The non-restoring algorithm is more practical: it avoids the restoration of the remainder allowing some extra computations in each iteration. This is done as follows:

$$S^{(i)} = \begin{cases} 2S^{(i-1)} - q_{n-i}(2^n D), \text{if } 2S^{(i-1)} \geq 0 \\ 2S^{(i-1)} + q_{n-i}(2^n D), \text{if } 2S^{(i-1)} < 0 \end{cases}, \quad (3)$$

with $S^{(0)} = Y$, $S^{(n)} = 2^n S$

and $q_{n-i} = \begin{cases} 1, \text{if } 2S^{(i-1)} \geq 0 \\ 0, \text{if } 2S^{(i-1)} < 0 \end{cases}$.

Speed-up of the division operation

Optimization of the division algorithms to speed-up the division operation is a laborious process. Nevertheless, some ways for this optimization exist. They are:

1. replacement of the divisor by its inverse value and following multiplication by the divider;
2. reducing the computation time of the partial remainder values speeding up the addition operations;
3. decreasing the number of addition operations during the computation of the partial remainder;
4. computing the quotient in residual number system (RNS).

Except the first one, the above approaches are in fact the modifications of the traditional restoring and non-restoring division algorithms.

One can significantly quicken the computation of the division if it will be changed to the fast multiplication: replace the divisor by its inverse value and multiply the dividend by *1/D*. Thus, the problem reduces to the effective determination of the value *1/D*. Commonly, this task is solved by one of three methods: using Taylor's series, Newton-Raffson approximation or using the properties of the odd numbers [2,3]. The application of the inverse method is typical for floating-point implementation of the division and is commonly used in the microprocessors.

A speed-up of the division operation is achieved by optimizing the computation of the partial remainder values. This is done using special constructions for the fast addition and carry propagation [2,4], e.g. matrix schemes that have a regular structure and are suitable for the implementation in programmable logic.

The most popular methods for the speed-up of the division operation are the modifications of the traditional restoring and non-restoring algorithms. Among these modifications, the SRT algorithm is the most widely used [5-7]. The main idea of this method is to use limited comparisons to speed-up the selection of the quotient. Typically, the hardware implementations of the different forms of the SRT algorithm present the lowest area requirements but, at the same time, the high latency is a drawback of these designs.

The *i*-th iteration in the SRT algorithm is expressed as

$$q_i = \begin{cases} 1, \text{if } 2S^{(i-1)} \geq D \\ 0, \text{if } -D \leq 2S^{(i-1)} < D \\ -1, \text{if } 2S^{(i-1)} < -D \end{cases} \quad (4)$$

$$S^{(i)} = 2S^{(i-1)} - q_i D. \quad (5)$$

In Equations (4) and (5) the quotient and remainder are presented in the non-binary number system; in this situation the values are {-1, 0, 1}. After all iterations if the remainder is negative, the remainder and the quotient have to be corrected. The last step in the algorithm is the conversion of the result to the binary system {0,1}, which can be done on the fly. It is necessary to mention that the computations in the residual number systems have a higher efficiency, e.g., as for the Booth multiplication algorithm [2,4]. At the same time, utilization of such technique leads to the complex design of the division modules: in particular, the logic that determines the operation type in each iteration. For this purpose different memories and look-up tables are used. Anyway, the speed-up has a higher priority and that is one of the main reasons why this technique is frequently used.

Fixed-point division core from Xilinx

Xilinx produces one of the standard solutions of the fixed-point dividers. It is a parameterized IP Core "Pipelined Divider" with the following features [8]:

- drop-in module for all Virtex, Virtex-II, Virtex-II Pro, Spartan-3, Virtex-4 etc. FPGAs;
- the dividend value can range from 1 to 32 bits;
- the divisor value can range from 3 to 32 bits;
- produces the quotient with integer or fractional remainder;
- the remainder value in fractional mode can range from 3 to 32 bits.

Main properties of this IP Core are given in Table 1. They are summarized from the implementations of the 32-bit IP Core on the Virtex-II Pro Xilinx FPGA. This core has obvious advantages like fully pipelined architecture, 32-bit operand widths. Nevertheless, it has some drawbacks: a large area occupied by the design and the limited widths of the operands and the fixed-point result of the division. The latter reduces the module's application field, i.e. if there is a need to increase the precision of the result one has to find some ways to apply different scaling techniques for the operands. This makes the utilization of the standard IP Core not universal for all tasks.

Table 1 Properties of the 32-bit "Pipelined Divider v.3.0" IP Core from Xilinx

Property	Number of bits in remainder		
	8	16	32
Number of slices	2247	2742	3843
Number of Flip-Flops	4020	4904	6864
Number of look-up tables	1400	1680	2240
F_{max}, MHz	204,3	201,6	193,1

Related work

Studies of efficient implementations of the integer division algorithms in the hardware are carried out for a long time. Works in field of semi-custom VLSI designs are presented in [1,6]. Complete correctness proofs of the RNS division algorithms are given in [1], whereas the first implementation in that study cannot be utilized for the computations of the high number of remainder bits and second implementation uses the weighted fractional numbers that reduces the overall performance. Another solution of the online integer divider is given in [9], and the implementation of the SRT algorithm is given in [5]. Nevertheless, some place for the improvement of the special-purpose division hardware is still left, e.g., implementing the division without additional pre-scaling as in [9]. In opposite to the previous work the current study is directed for fast computation of the high-precision integer division (for big remainders with 64, 128 or more bits) and possible optimization of the logic resources of the FPGAs.

3. IMPLEMENTATION OF HIGH-SPEED DIVISION MODULES

Fixed-point division modules on FPGA

The restoring, non-restoring and SRT division algorithms were selected for the implementation on the FPGA basis. All algorithms were analyzed in order to define the computational structure and were coded in VHDL. This coding was done in such a way that no special internal constructions of Xilinx FPGAs like adders, multipliers, look-up tables etc. were utilized explicitly. Such type of designs can be used in non-Xilinx programmable logic devices like Altera.

All designs have the same interface including input busses of the *n*-bit wide divided and *m*-bit wide divisor, output *r*-bit wide quotient and *q*-bit wide remainder busses and a control bus. Signals on the control bus are "load", "start", "done" and "error". The designs are fully synchronous and parameterized: all widths are declared as *generic* parameters of the VHDL modules. This parameterization allows selecting an arbitrary large bit-widths and an arbitrary precision of the division result, respectively. In addition, the number of clock cycles required to compute the remainder is directly proportional to the number of desired bits plus number of initialization cycles.

In fact, the implementation of the modules is direct coding of the algorithms in VHDL. Much attention was paid to the analysis of all algorithms in order to define the effective computational structures and to pipeline these structures. In each module the increase of the precision (number of remainder bits) was achieved not by the expansion of the dividend width but using the certain number of iterations of the division algorithm. This approach has an advantage over the increasing sizes of the registers, adders and multipliers.

Besides the pipelining, big improvement in speed of the designs was achieved using the Xilinx software-specific timing and area constraints. Detailed selection of the internal nets for the application of timing constraints had significant impact on the design speed. The same was with the area constraints: all designs were under "hand-based" optimization. All division modules were implemented on Virtex-II Pro FPGAs using the Xilinx ISE 6.3i software packet. All steps, e.g. XST synthesis, place-and-route, were executed with the highest optimization settings, that had significant impact on the design speed when the precision of the remainder was more than 64 bits.

Verification of the implemented division modules was done using the ModelSIM XE 5.8 software. Each design was simulated using the prepared test-bench with more than several hundreds different test combinations of the input operands. Results of the each division operation were saved to intermediate file and checked using script with the predefined test data. These simulations were carried out after XST synthesis in order to check the logic, and after the place-and-route operation in order to check the delays and correctness of the routed design.

Implementation results

The main aim of the study was to implement the fixed-point division modules with an arbitrary (unbounded) remainder precision in FPGA, to optimize and to test them. Moreover, the precision must exceed the precision of the standard IP Core "Pipelined Divider" from Xilinx. The parameterized designs were implemented on Virtex-II Pro Xilinx FPGAs with output remainder widths from q=8 to q=128 bits for different widths of the input operands. The complete set of statistical information was gathered for detailed analyses. The following parameters of the designs were selected: the maximum design frequency, the number of utilized slices, the number of look-up tables, the number of flip-flops, input-output ports etc. All these data was gathered for different combinations (n,m,r,q) of input and output widths of the operands.

The results for the 32-bit input operands of the division modules compared to the 32-bit IP Core "Pipelined Divider" from Xilinx are presented in Figures 1-3. The implementation of non-restoring algorithm significantly exceeds the Xilinx division core in speed and has noticeably smaller size.

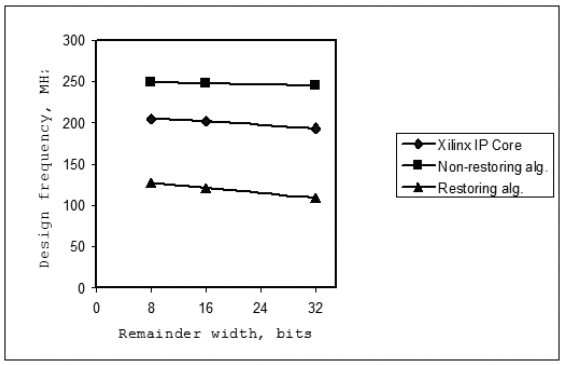

Figure 1. Frequency of the design vs. remainder widths. All inputs are 32-bit wide.

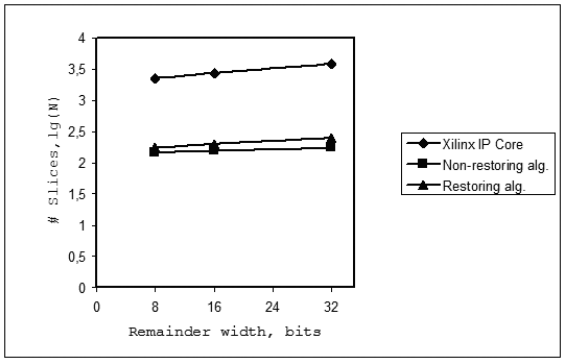

Figure 2. Number of utilized slices vs. remainder width. All inputs are 32-bit wide.

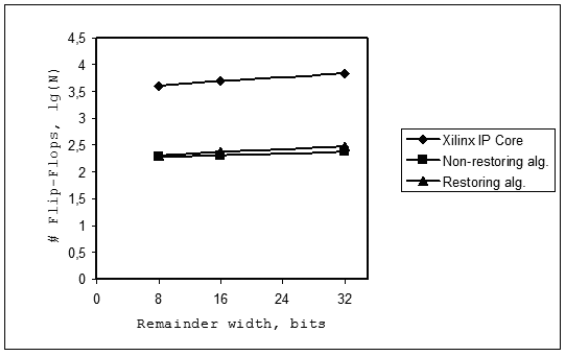

Figure 3. Number of utilized Flip-Flops vs. remainder width. All inputs are 32-bit wide.

The experimental results for the high-precision designs are combined in Tables 2 – 4. The frequency of each module is presented for different sizes of the input operands and the precision of the result. Also the time of computation is given for each case. For the IP Core from Xilinx number of cycles for the division is equal to the $N = n + q + 3$, whereas for designs in this study equal to $N \le n + q + 1$ for all algorithms, because number of cycles, required for the division depends on the input numbers (it is faster for the numbers that can be expressed with the number of bits smaller than *n* and *m*).

The presented results clearly show the superiority of the non-restoring algorithm for the high-speed applications that require the precise division results. It has to be mentioned, that the design frequency is changed slightly with the increase of the widths *n* and *m* of the input operands.

Table 2 Frequency (in MHz) / Time of computation (in ns) of different division modules with 32-bit wide input operands

Design \ Remainder, bits	8	16	32	64	128
Xilinx IP Core	204,3 / 211	201,6 / 253	193,1 / 350	-	-
SRT algorithm	78,1 / 525	77,2 / 635	76,1 / 854	72,4 / 1340	64,3 / 2504
Restoring algorithm	126,6 / 324	120,9 / 405	108,8 / 597	89,1 / 1089	81,2 / 1983
Non-restoring algorithm	248,6 / 165	247,4 / 198	244,9 / 265	193,1 / 502	180,3 / 893

Table 3 Frequency (in MHz) / Time of computation (in ns) of different division modules with 64-bit wide input operands

Design \ Remainder, bits	32	64	128
Xilinx IP Core	-	-	-
SRT algorithm	62,5 / 1552	61,8 / 2087	60,5 / 3190
Restoring algorithm	75,3 / 1288	57,8 / 2232	48,4 / 3988
Non-restoring algorithm	173,6 / 559	160,3 / 805	157,6 / 1225

Table 4 Frequency (in MHz) / Time of computation (in ns) of different division modules with 128-bit wide input operands

Design \ Remainder, bits	32	64	128
Xilinx IP Core	-	-	-
SRT algorithm	47,9 / 3361	47,1 / 4098	46,2 / 5563
Restoring algorithm	42,8 / 3762	38,9 / 4961	36,6 / 7022
Non-restoring algorithm	92,2 / 1746	89,9 / 2147	86,5 / 2971

4. CONCLUSION

A lot of computational tasks implemented in programmable logic use the fixed-point division operation, whose preciseness often has to be extended. The standard IP Core "Pipelined Divider" from Xilinx has a limitation – the widths of the operands and precision of the result can be at most 32 bits. The detailed study of different division algorithms has shown the advantages of the implementation of the non-restoring algorithm versus other ones. Moreover, our design is completely parameterized and there are no restrictions for the widths of the input operands and the result of division. For the 32-bit wide dividend, divider, quotient and remainder the division module with non-restoring algorithm has the clock frequency 245 MHz vs. 193 MHz of IP Core from Xilinx on Virtex-II Pro FPGA. Our fixed-point division modules were tested for the large widths of input operands, e.g. 64-and 128-bit, and for the different precision values of the result.

5. REFERENCES

1. A.A. Hiasat, H.S. Abdel-Aty-Zohdy, "Semi-Custom VLSI Design and Implementation of a New Efficient RNS Division Algorithm", The Computer Journal, Vol. 42, No. 3, 1999, pp. 232-240.
2. S.M. Mueller, W.J. Paul, Computer Architecture Complexity and Correctness: Springer-Verlag, 2000.
3. Parhami Behrooz, Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press: New York, 2000.
4. W.J. Paul, P.-M. Seidel, "To Booth or not to Booth", Integration, the VLSI journal, Vol. 32, No. 1-3, 2002, pp. 5-40.
5. D.L. Harris, S.F. Oberman, M.A. Horowitz, "SRT Division Architectures and Implementations", Proceedings of 13th IEEE International Symposium on Computer Arithmetic, 1997, pp. 18-25.
6. M.D. Ercegovac, T. Lang, Division and Square-Root Algorithms: Digit-Recurrence Algorithms and Implementations. – Norwell, MA: Kluwer Academic Publishers, 1994.
7. S.F. Oberman, M. Flynn, "Division algorithms and implementations", IEEE Transactions on Computers. Vol. 46, No. 8, 1997. pp. 833–854.
8. Xilinx Inc. Pipelined Divider V3. Product Specification. 2004.
9. A.F. Tenca, M.D. Ercegovac, "On the Design of High-Radix On-Line Division for Long Precision", Proceedings of the 14th IEEE Symposium on Computer Arithmetic, 1999, pp. 44-51.

Received: Nov 2005. Accepted: Feb. 2006.