Design and Implementation of an Optimized Double Precision Floating Point Divider on FPGA

Shamna.K[1] and S.R Ramesh[2]

[1]Student , M-Tech VLSI Design , [2]Assistant. Professor
Dept. of Electronics and Communication Engineering
Amrita Vishwa Vidyapeetham, Coimbatore, India
[1]shamnakoleri@yahoo.co.in , [2]sr_ramesh@cb.amrita.edu

Abstract

Floating-point division is generally regarded as a low frequency, high latency operation in typical floating-point applications.So due to this not much development had taken place in this field. But nowadays floating point divider has become indispensable and increasingly important in many modern applications. Most of the previous implementation required much larger area and latencies. In this paper an area optimized design and implementation of a sequential and pipelined double precision floating point divider is presented. This design is then mapped onto an FPGA chip without utilizing any of its embedded features

Keywords: *Double precision, Floating point unit, divider, FPGA.*

1. Introduction

Modern applications comprise several floating point operations like addition, multiplication, division, and square root etc. In recent FPUs, emphasis has been placed on designing ever-faster adders and multipliers, with division receiving less attention. The typical range for addition latency is two to four machine cycles and the range for multiplication is two to eight machine cycles. In contrast, the latency for double precision division ranges from six to 61 cycles and square root is often far larger. Most emphasis has been placed on improving the performance of addition and multiplication. As the performance gap widened between these operations and division, floating-point algorithms and applications have been slowly rewritten to account for this gap by mitigating the use of division. Thus current applications and benchmarks are usually written assuming that division is an inherently slow operation and should be used sparingly. Thus division was considered as a 'black art' among system designers.

But with the advent of new technologies a new algorithm for the efficient implementation of division also became necessary. As such many algorithms were developed for divider which includes subtractive method, functional iterations which uses multipliers and algorithms for faster computation of division like high radix algorithm. But most of these algorithms namely functional iteration and high radix algorithm required multipliers and thus consumed large area and power. But large area for division alone is not desirable. So digit recurrence algorithm which uses subtractive method for computation could be used as it consumes much less area when compared with other algorithms.

So we have designed a sequential double precision floating point divider to achieve a low area with moderate latency. The throughput can be increased by pipelining the

designed unit. This design unit is mapped onto FPGA in order to achieve higher data rates. This design is implemented in Cyclone II FPGA and it is seen that our design requires only less area and works with moderate latency.

The double precision floating point divider presented here is based on IEEE 754 binary floating point standard. Having a standard ensures that all compliant machines will produce the same outputs for the same program. The standard is very complex and difficult to implement efficiently.

2. Previous work

Formerly division was less frequently used and so no much development had taken place in its field. But with the advent of new technology floating point computation also became important and was widely used. Thus implementation efficiency of addition and multiplication were much developed. But the division stood back [7]. So the performance of the system that used floating point divider was greatly affected [8].

So a new algorithm for efficient implementation of division also became necessary. As such many algorithms were put forth [9]. Functional iteration used multipliers for computation and hence they required larger area but they required only less latency. On the other hand digit recurrence required small area [11] but latency had to be compromised. But latency can be reduced by increasing the radix.

The divider can be implemented in many ways in order to achieve low area, low latency and high throughput.[6] [10]

The throughput can be increased by partial unrolling of the dividing unit and inserting pipeline registers in between the dividing unit [3], [4].Then a library of floating point can be developed for FPGAs according to compliance with IEEE [5].

3. Double precision floating point divider based on IEEE 754 binary floating point standard

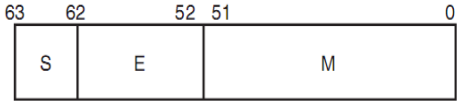


Fig.1 The Double Precision Format

Floating point divider relies on IEEE 754 binary floating point standard. The standard specifies different types of precision. We represent a binary floating-point number with three fields: a sign bit s, an exponent field e and a fraction field f. According to this standard a double precision floating point number(N) is 64 bit width consisting of a sign bit(S), 11 bit exponent(E) and 52 bit mantissa(M). It can be represented as $N= (-1)^{s}.2^{e}.S$; Where S is the significand (fractional) part and can be represented as 1.f, where f is the fractional part and 1 is the hidden bit. e = E – E bias; where E bias = 1023;

For double precision numbers, the range of the unbiased exponent e is [-1022, 1023], which translates to a range of [1,2046] for the biased exponent E. The values E=0 and E=2047 are reserved for special quantities. The number zero is represented with E=0 and f=0.The hidden significand bit is also 0 and not 1. Zero has a positive or negative sign like normal numbers. When E=0 and $f \neq 0$ then the number has e=-1022 and a

significand S=0.f. The hidden bit is 0 and not 1 and the sign is determined as for normal numbers. Such numbers are referred to as "denormalised".

An exponent E=2047 and a fraction f=0 represent infinity. The sign of infinity is determined as for normal numbers.Finally, an exponent E=2047 and a fraction f≠ zero represent the symbolic unsigned entity NaN (Not a Number), which is produced by operations like 0/0 and 0/∞ .The standard does not specify any NaN values, allowing the implementation of multiple NaN. Here only one NaN is provided with E=2047 and f =0.00001.The IEEE standard specifies four rounding modes. Here we are using only rounding to the nearest mode.

4. Double precision floating point divider architecture

The divider receives two 64 bit floating point numbers. First these numbers are unpacked by separating the numbers into sign bit, exponent bits and mantissa bits as shown in fig 1.

The sign logic is a simple XOR. The exponents of the two numbers are subtracted and then added with a bias number i.e., 1023. Mantissa division block performs division using digit recurrence algorithm. It takes more than 55 clock cycles. After this the output of mantissa division is normalised i.e., if the MSB is of the result obtained is not 1, then it is left shifted to make the MSB 1. If changes are made by shifting then corresponding changes has to be made in exponent also.

After mantissa division the output is 55 bit long. But we require only 53 bit mantissa. So after normalization the 55 bit output is passed on to the rounding control. Here rounding decision is made based on the last 2 bits of the LSB. They are the guard bit and the sticky bit respectively. From these 2 bits and other lower bits an additional bit called round bit is calculated. This bit decides whether rounding has to be performed or not. If the round bit is 1, then a 1 has to be added to the LSB of the output and then scaled to 53 bits. These functions are performed in the rounding block according to the decision taken in the rounding control block.

If a 1 is added to the LSB of the mantissa then corresponding changes has to be made in the exponent part also. This is carried out in the exponent adjustment block.

Finally the output from the Sign block, Exponent adjustment block and the Rounding block are concatenated in the packing block to produce the final quotient. The whole circuit takes about 62 clock cycles.

5. Pipelining of Double precision floating point divider

For increasing the throughput of the circuit the division step is unrolled as shown in Fig 3 to produce a combinational circuit. Then pipeline latches can be inserted in between as depicted in fig. 4 in order to increase the throughput.

But pipelining in this way causes much area overhead. So in order to minimise area partial unrolling of the circuit could be done. Partial unrolling of the can be done by unrolling the circuit to 2,4,8,16 or 28 stages and inserting the pipeline registers after each stage. By doing so, the throughput can be increased without much area overhead.

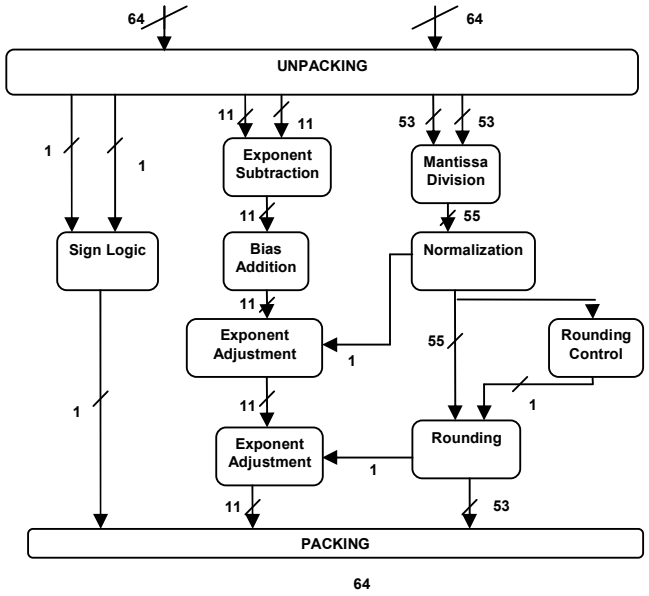
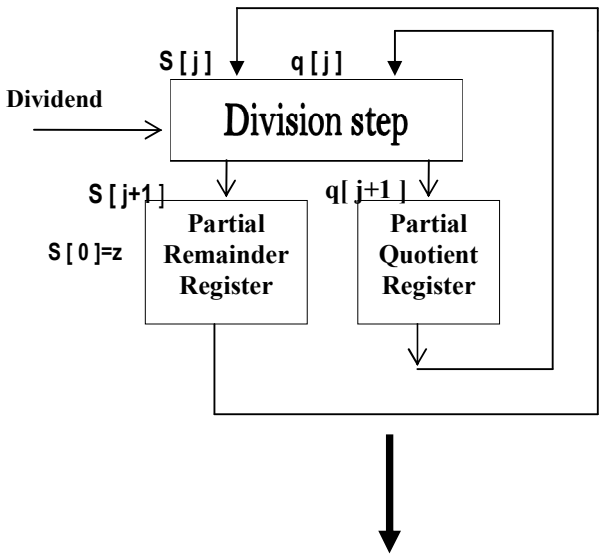


Fig.2 The divider block diagram

The area of a pipeline design can be expressed as

$$Apipe = nc + \lceil n/m \rceil r \quad (1)$$

where c is the combinational area of a single iteration, r is the number of bit registers required for a single pipeline stages, d is the execution delay of a single iteration and n is the number of iterations in the sequential design.



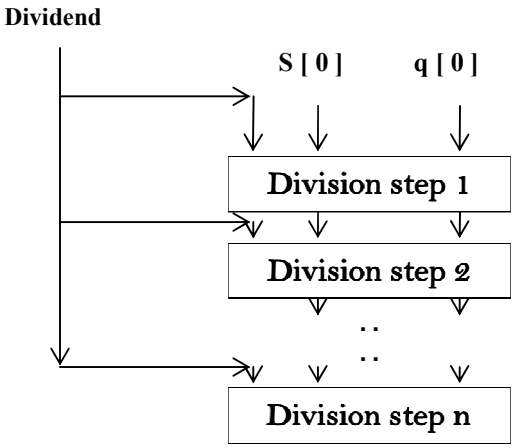


Fig.3 Unrolling steps of the division hardware

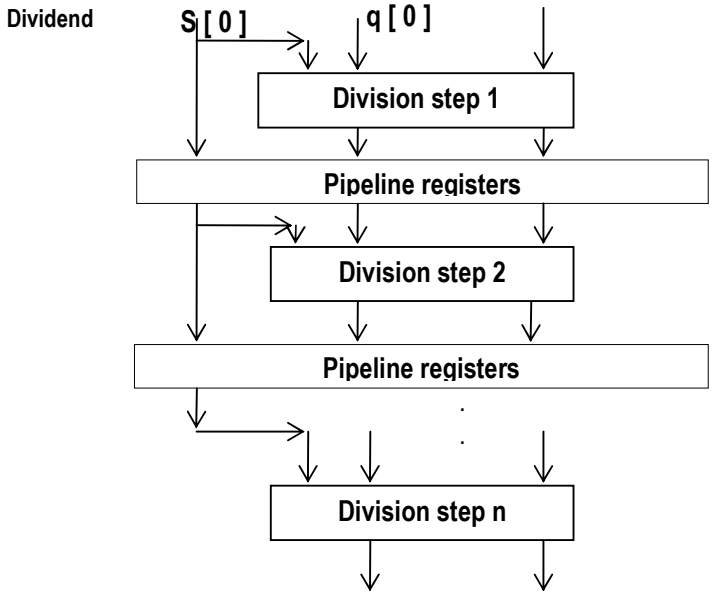


Fig.4 Inserting pipeline latches in the divider circuit

6. Implementation results

The divider circuit based on digit recurrence algorithm was simulated in Modelsim 6.4c and synthesized in Altera Quartus II version 9 which was mapped on to Cyclone II FPGA.

Table 1 Area utilized by all the modules in double precision floating point divider

Modules	Total registers		Total combinational functions		Total logic elements	
	No. Of elements	% Area	No. Of elements	% Area	No. Of elements	% Area
Unpacking	128/18752	<1	128/18752	<1	128/18752	<1
Sign	1/18752	<1	1/18752	<1	1/18752	<1
Exponent	11/18752	<1	22/18752	<1	22/18752	<1
Division	55/18752	<1	8423/18752	45	8423/18752	45
Normalization with exponent adjustment	131/18752	<1	102/18752	<1	156/18752	<1
Rounding with exponent adjustment	64/18752	<1	146/18752	<1	146/18752	<1
Packing	64/18752	<1	64/18752	<1	64/18752	<1

Table 2 Power utilized by all the modules in double precision floating point divider

Module	Static power dissipation (mW)	Dynamic power dissipation (mW)	I/O thermal power dissipation(mW)
Unpacking	47.39	6.79	36.2
Sign	47.35	.17	20.73
Exponent	47.36	2.53	24.26
Division	47.51	11.03	40.74
Normalization with exponent adjustment	47.39	5.42	35.86
Rounding with exponent adjustment	47.39	4.63	35.79
Packing	47.42	9.4	36.3

Table 3 Area utilized by a double precision floating point divider using digit recurrence algorithm.

Component	% Area
Total logic registers	2
Total combinational functions	49
Total logic elements	49

Table 4 Power dissipated by a double precision floating point divider using digit recurrence algorithm

Static power dissipation (mW)	47.41
Dynamic power dissipation (mW)	82.91
I/O thermal power dissipation (mW)	52.16
Clock frequency	265MHz

Table 5 Comparison between digit recurrence algorithm and functional iteration algorithm

	Digit Recurrence Algorithm	Functional Iteration Algorithm
Components	**% Area**	**% Area**
Total logic elements	49	54
Total combinational functions	49	54
Embedded Multiplier	0	98

From the table 5 it is evident that the digit recurrence algorithm requires only small area when compared with functional iteration algorithm. So pipelining of these units does not produce much area overhead than other division algorithms.

7. Future enhancement

The latency of the divider can be reduced by using a secondary clock for mantissa division alone. The frequency of the secondary clock is twice larger than the primary clock. The primary clock is applied to all other parts of the divider unit. This is done because mantissa division is the slowest part and it requires more than 55 clock cycles for mantissa computation.

The latency can also be reduced by using a cache memory which can be used to store the quotient values of the data with high probability of occurrence. By doing so the latency can be reduced up to 6 clock cycles.

An asynchronous double precision floating point divider can be designed for reusability of the divider unit in various systems operating at different frequency. Also power consumption can be reduced to a great extend as the global clock is removed and also clock skew problem also can be reduced by designing in this manner.

8. Conclusion

This paper presents the iterative and pipelined designs of double precision floating point divider unit. The design presented here can produce performances that are comparable to, and in some case higher than, non-iterative designs based on number representations of higher radices. The iterative design of the divider requires less area. Since the pipelining of our iterative designs is intended to accelerate compute-intensive applications on FPGA chips, full unrolling of these iterative designs is highly desirable since it can produce maximum performance. But it cause significant area overhead. So partial unrolling of the divider design is done without affecting the performance of the divider.

9. References

[1] K. Scott Hemmert and Keith D. Underwood “Floating Point Divider Design for FPGAs”, IEEE Transaction on very large scale integration systems,vol. 15, No. 1, pp. 115-118,Jan 2007.
[2] Mohamed anane, Hamid Bessalah ,Mohamed Issad, Nadjia Anane and Hassen Salhi “Higher radix and redundancy factor for floating point SRT Division”, IEEE Transaction on very large scale integration systems, vol. 16, no. 16, pp. 122-128,June 2008.
[3] Anuja Jayraj Thakkar and Abdel Ejnioui “ Pipelining of Double Precision Floating Point Divider and Square Root Operations ,” Proceedings of the 44th annual southeast regional conference, March 2006.
[4] Anuja Jayraj Thakkar and Abdel Ejnioui “ Design and Implementation of Double Precision Floating Point Divider And Square Root Operations On FPGAs ,”IEEE Conference on field programmable technology,2006
[5] Govindu G ,Scrofano.R and Prasannna V.K “ A Library of Parameterizable Floating Point Cores for FPGAs and their Application to Scientific Computing,” International Conference on engineering of reconfigurable systems and algorithms,2005.
[6] X .Wang and B.E Nelson, "Tradeoffs of designing floating point division and square root on virtex fpgas”, International Conference on engineering of reconfigurable systems and algorithms,2004.
[7] S. Paschalakis and P. Lee ,”Double precision Floating point arithmetic on fpgas "IEEE Conference on field programmable technology,2003.
[8] S.F. Oberman and M.J. Flynn, “Design Issues in Division and Other Floating-Point Operations,” IEEE Trans. Computers, vol. 46,no. 2, pp. 154-161, Feb. 1997.
[9] S.F. Oberman and M.J. Flynn, “Division Algorithms and Implementations,” IEEE Trans. Computers, vol. 46, no. 8, pp. 833-854, Aug. 1997.
[10] Peter Soderquist, Miriam Leeser, "Division and Square Root: Choosing the Right Implementation," IEEE Micro, vol. 17, no. 4, pp. 56-66, July/Aug. 1997.
[11] Synthesis of arithmetic circuits-FPGA, ASIC, and embedded systems by Jean-Pierre Deschamps Gery Jean Antoine Bioul and Gustavo D. Sutter, A John Wiley & Sons, inc., publication