An FPGA Based High Performance IEEE - 754 Digit Recurrence Floating Point Double Precision Divisor Using Verilog

Purna Ramesh Addanki [1], Venkata Nagaratna Tilak Alapati[2]
and Mallikarjuna Prasad Avana[3]

[1]Department of ECE, Sri Vasavi Engineering College, Pedatadepalli, Tadepalligudem, India

[2] Professor of ECE, V.R.Siddhartha Engineering College, Kanuru, Vijayawada, India

[3]Professor of ECE, JNTUK, Kakinada, India

purnarameshaddanki@gmail.com, avntilak@yahoo.com, a_malli65@yahoo.com

Abstract

Current Floating-point divisor architectures have low frequency, larger area and high latency in nature. With advent of more graphic, scientific and medical applications, floating point dividers have become indispensable and increasingly important. However, most of these modern applications need higher frequency or low latency of operations with minimal area occupancy. In this work, highly optimized pipelined architecture of an IEEE-754 standard double precision floating point divider is designed to achieve high frequency on FPGAs. By using secondary clock to perform mantissa division the overall latency of the divisor is reduced to 30 clock cycles, i.e. 52% less compared to conventional divisors. This design is mapped onto a Virtex-6 FPGA and an operating frequency of 452.69 MHz is achieved. The proposed design also handles all the IEEE specified four rounding modes, overflow, underflow and various exception conditions.

Keywords: *Double precision, floating point unit, divider, fpga, IEEE-754*

1. Introduction

Floating point arithmetic is widely used in many scientific and signal processing applications. Implementing arithmetic operations for floating point numbers in hardware is very challenging. Among the operations (add, subtract, multiply, divide), division is generally the most difficult to implement in hardware. In recent floating point units (FPUs), the designer's concentration has been placed more on designing ever-faster adders and multipliers compared to division. 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 about 61 cycles and square root is often far larger. 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 [1].

Efficiency of addition and multiplication were much developed but division stood back [2] and the performance of the system that used floating point divider was greatly affected [7]. Formerly division was less frequently used and hence not much development had taken place in its field. But with the advent of new technology applications floating point division also became important. Therefore a new algorithm for efficient implementation of division also became necessary. As such many algorithms (functional iteration, very high radix, table look-up & variable latency) were put forth [3]. The throughput of a divider can be increased by using a high radix SRT algorithm [8] and add-multiply infrastructure [4].

The challenge in FPGAs is a right trade-off between clock speed, latency, throughput, and area [5]. Double precision floating point divider can be implemented based on SRT division algorithm. This algorithm depends on the radix and the redundancy factor. At each iteration, the SRT algorithm performs a multiplication by the quotient digit. So at each iteration SRT needs a multiplier. To overcome this quotient digit is decomposing into two or three terms multiples of 2. Radix-8 with a maximum redundancy factor gives the best performance [6].

Double precision floating point divider can be implemented based on partial and full unrolling of the iterations in low radix digit recurrence and inserting pipeline registers in between the dividing unit results in increasing the throughput [9, 10].

With advent of more graphic, scientific and medical applications, floating point division has become indispensable and increasingly important. However, most of these modern applications need higher frequency or low latency of operations with minimal area occupancy. As such many algorithms were developed for divider which includes binomial expansion [11].

Subtractive method and functional iterations uses multipliers and algorithms for faster computation of division like high radix algorithm. But most of these algorithms require multipliers and thus consumed large area and power. The digit recurrence algorithm [13] which uses subtractive method for computation could be used as it consumes much less area when compared with other algorithms.

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.

We have designed a digit recurrence double precision floating point divider with secondary clock to calculate mantissa so as to achieve a low latency. Also, we have incorporated more pipeline stages to achieve high frequency and throughput. The design is implemented in Xilinx Virtex-6 FPGA and it is verified that this design requires minimal area and also it operates at a very high frequency of 452.69MHz compared to a frequency of 100.70 MHz using methods like non-iterative designs based on high radix numbers, sequential and pipelined designs [10].

2. Double Precision Floating Point Divider Based on IEEE-754 Binary Floating Point Standard

Floating point divider relies on IEEE-754 binary floating point standard. The IEEE-754 standard defines how double precision floating point numbers are represented. 64 bits are used to represent this number. The double precision floating point format is shown in Figure 1.

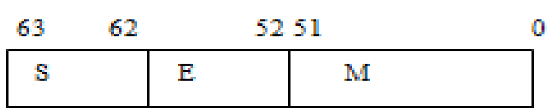


Figure 1. The Double Precision Floating Point Format

The sign bit occupies bit 63. ‘1’ signifies a negative number and ‘0’ a positive number. The exponent field is 11 bits long, occupying bits 62-52. The value in this 11-bit field is offset by 1023, so the actual exponent used to calculate the value of the number is 2^(e-1023). The mantissa is 52 bits long and occupies bits 51-0. There is a leading '1' that is not included in the mantissa, but it is part of the value of the number for all double precision floating point numbers with a value in the exponent field greater than 0. A 0 in the exponent field corresponds to a de-normalized number, which is explained in the next section. The actual value of the double precision floating point number is

$$\text{Value} = -1\text{\^{}}(\text{sign bit}) * 2\text{\^{}}(\text{exponent} - 1023) * 1.(\text{mantissa})$$

The IEEE standard specifies four rounding modes; round to nearest, round to zero, round to positive infinity, and round to negative infinity. The representation of the special cases in floating point double precision numbers is shown in Figure 2.

Positive Infinity is:

Sign	Exponent	Mantissa
63	62..........52	51..0
0	11111111111	0000000000000000000000000000000000000000000000000000

Negative Infinity is:

Sign	Exponent	Mantissa
63	62..........52	51..0
1	11111111111	0000000000000000000000000000000000000000000000000000

Quiet Not a Number (QNaN) is:

Sign	Exponent	Mantissa
63	62..........52	51..0
x	11111111111	1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Signaling Not a Number (SNaN) is:

Sign	Exponent	Mantissa
63	62..........52	51..0
x	11111111111	0xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Figure 2. Special Cases in Representing Floating Point Double Precision Numbers

3. Proposed Architecture

3.1. Base Architecture for Divider

The divider receives two 64-bit floating point numbers. First these numbers are unpacked by separating the numbers into sign, exponent, and mantissa bits. 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 normalized, i.e., if the MSB 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 [1].

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 mode selected by the user. This mode decides whether rounding has to be performed - round to nearest (code = 00), round to zero (code = 01), round to positive infinity (code = 10), and round to negative infinity (code = 11). Based on the rounding changes to the mantissa corresponding changes has to be made in the exponent part also.

For round to nearest mode, if the first extra remainder bit is a '1', and the LSB of the mantissa is a '1', then this will trigger rounding. For round to zero mode, no rounding is performed, unless the output is positive or negative infinity. This is due to how each operation is performed. For multiply and divide, the remainder is left of the mantissa, and so in essence, the operation is already rounding to zero even before the result of the operation is passed to the rounding module. For round to positive infinity mode, the two extra remainder bits are checked, and if there is a '1' in either bit, or the sign bit is '0', then the rounding amount will be triggered. Likewise, for round to negative infinity mode, the two extra remainder bits are checked, and if there is a '1' in both bits, and the sign bit is '1', then the rounding amount will be triggered.

Normalized mantissa will be checked for any exceptions, where all of the special cases are checked. The special cases are

1. Divide by 0 – result is infinity, positive or negative, depending on the sign of operand A.
2. Divide 0 by 0 – result is SNaN, and the invalid signal will be asserted.
3. Divide infinity by infinity - result is SNaN, and the invalid signal will be asserted.
4. Divide by infinity – result is 0, positive or negative, depending on the sign of operand A and the underflow signal will be asserted.
5. Divide overflow – result is infinity, and the overflow signal will be asserted.
6. Divide underflow – result is 0, and the underflow signal will be asserted.
7. One or both inputs are QNaN – output is QNaN.
8. One or both inputs are SNaN – output is QNaN, and the invalid signal will be asserted.

If any of the above cases occurs, the exception signal will be asserted. If the output is positive infinity, and the rounding mode is round to zero or round to negative infinity, then the output will be rounded down to the largest positive number (exponent = 2046 and mantissa is all 1's). Likewise, if the output is negative infinity, and the rounding mode is round to zero or round to positive infinity, then the output will be rounded down to the largest negative number. The rounding of infinity occurs in the exceptions module, not in the rounding module.

QNaN is defined as Quiet Not a Number. SNaN is defined as Signaling Not a Number. If either input is a SNaN, then the operation is invalid. The output in that case will be a QNaN. For all other invalid operations, the output will be a SNaN. If either input is a QNaN, the operation will not be performed, and the output will be a QNaN. If both inputs are QNaNs, the output will be the QNaN in operand A. The use of Not a Number is consistent with the IEEE-754 standard.

Finally all the outputs from the sign, exponent and mantissa are concatenated to produce the final quotient. The whole operation takes about 62 clock cycles.

3.2. Reducing the Latency using Secondary Clock

The latency of the divider is 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. So, using double the clock frequency for mantissa calculation effectively reduces the overall latency of the divider to 30 cycles.

3.3 Increasing the Frequency of Divider using Pipelining

For increasing the frequency or throughput of the circuit the division step is unrolled and then several pipelining stages are inserted in between each minor operation.

The area of a pipeline design can be expressed as [1] $A_{pipe} = nc + [n/m]r$

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

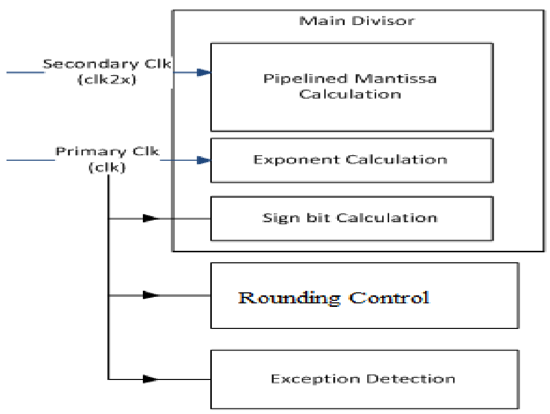


Figure 3. Proposed Architecture for Floating Point Double Precision Divisor

The final proposed architecture with secondary clock and pipelining stages is shown in Figure 3.The Figure 4 shows the black box view of floating point double precision divisor.

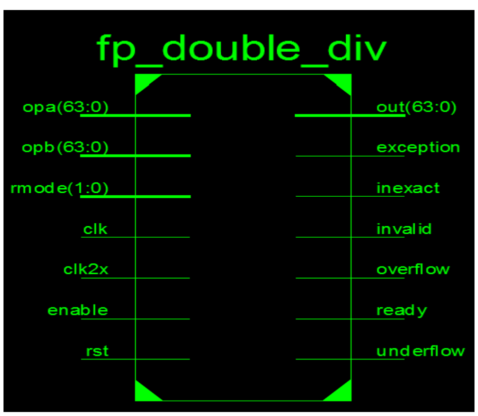


Figure 4. Black Box View of Floating Point Double Precision Divisor

4. Results

The divider circuit based on digit recurrence algorithm was simulated in Modelsim 6.6c and synthesized using Xilinx ISE 13.1i which was mapped on to Virtex-6 FPGA. The simulation results of 64-bit floating point double precision divisor are shown in figure 5. The ‘opa’ and ‘opb’ are the inputs and ‘out’ is the output. The figure 6 gives the timing summary which indicates the operating frequency of 452.69MHz. Table 1 summarizes the device utilization for implementing the circuit on Virtex-6 FPGA. The number of slices required is 841. Table 2 gives the comparison of existing method [1] and the proposed method in terms of latency and operating frequency.

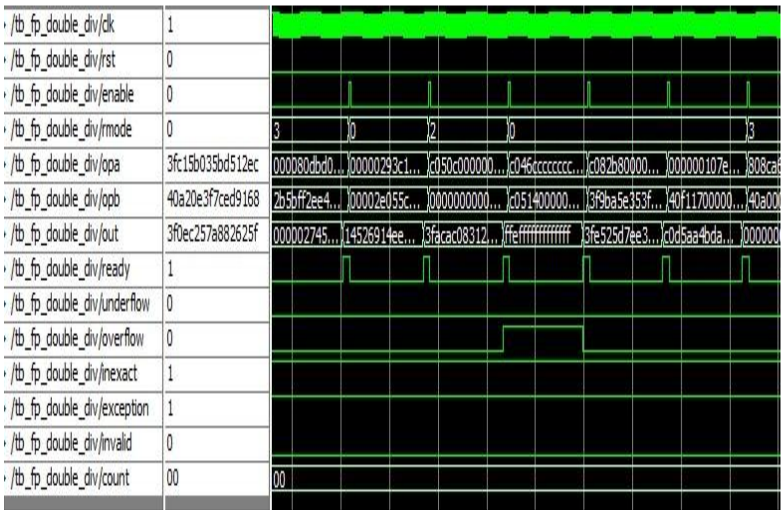


Figure 5. Simulation Results of Floating Point Double Precision Divisor

```
Timing errors: 0  Score: 0  (Setup/Max: 0, Hold: 0)

Constraints cover 9158 paths, 0 nets, and 3262 connections

Design statistics:
   Minimum period:   2.209ns{1}   (Maximum frequency: 452.694MHz)
```

Figure 6. Timing Summary of Floating Point Double Precision Divisor

Table 1. Device Utilization Summary (Selected device 6vlx75tff484-3) of Floating Point Double Precision Divisor

Slice Logic Utilization	Used
Number of Slice Registers(Flip-Flops)	1,992
Number of Slice LUTs	2,211
Number of occupied slices	**841**
Number of bonded IOBs	204

Table 2. Performance Comparison of Floating Point Double Precision Divisor

Parameter	Existing Method [1]	Proposed Method
No. of slices required	**678**	**841**
Frequency (MHz)	**265**	**452.69**
Latency (Clock cycles)	**53**	**30**

5. Conclusion

This paper presents the enhanced version of digit recurrence algorithm which offers 52% and 44 % less latency compared to conventional divisors and existing method [1] respectively. It can also be operated at a higher frequency of 452.69MHz. The design presented here can produce better performance as compared to 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 design is intended to accelerate compute-intensive applications on FPGA chips, full unrolling of these designs is highly desirable to achieve maximum performance.

The latency can be further reduced by using a cache (block) 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 [1].

An asynchronous double precision floating point divider can be designed for reusability of the divider unit in various systems operating at different frequencies. Also power consumption and clock skew problem can be reduced by removing the global clock.

References

[1] K. Shamna and S. R. Ramesh, "Design and Implementation of an Optimized Double Precision Floating Point Divider on FPGA", International Journal of Advanced Science and Technology, vol. 18, **(2010)** May, pp. 41-48.

[2] S. F. Oberman and M. J. Flynn, "Design Issues in Division and Other Floating-Point Operations", IEEE Trans. Computers, vol. 46, no. 2, **(1997)** February, pp. 154-161.

[3] S. F. Oberman and M. J. Flynn, "Division Algorithms and Implementations", IEEE Trans. Computers, vol. 46, no. 8, **(1997)** August, pp. 833-854.

[4] P. Soderquist and M. Leeser, "Division and Square Root: Choosing the Right Implementation", IEEE Micro, vol. 17, no. 4, **(1997)** July/August, pp. 56-66.

[5] S. Hemmert and K. D. Underwood, "Floating Point Divider Design for FPGAs", IEEE Transaction on very large scale integration systems, vol. 15, no. 1, **(2007)** January, pp. 115-118.

[6] M. Anane, H. Bessalah, M. Issad, N. Anane and H. Salhi, "Higher radix and redundancy factor for floating point SRT Division", IEEE Transaction on very large scale integration systems, vol. 16, no. 16, **(2008)** June, pp. 122-128.

[7] S. Paschalakis and P. Lee, "Double precision Floating Point Arithmetic on FPGAs", Proc. of IEEE Conference on Field Programmable Technology, **(2003)**, pp. 352-358.

[8] X. Wang and B. E. Nelson, "Tradeoffs of Designing Floating Point Division and Square Root on Virtex FPGAs", Proc. of the 11th Annual IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'03), **(2003)**.

[9] A. J. Thakkar and A. Ejnioui, "Pipelining of Double Precision Floating Point Divider and Square Root Operations", Proc. of the 44th Annual Southeast Regional Conference, **(2006)** March.

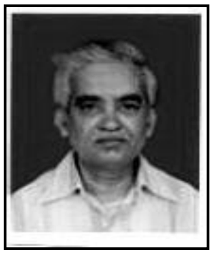
[10] A. J. Thakkar and A. Ejnioui, "Design and Implementation of Double Precision Floating Point Divider and Square Root Operations on FPGAs", Proc. of IEEE Conference on Field Programmable Technology, **(2006)**.

[11] M. K. Jaiswal, N. C. Choodan, "Efficient Implementation of Floating-Point Reciprocator on FPGA", 22nd International Conference on VLSI Design, **(2009)**, pp. 267-271.

[12] J. -P. Deschamps, G. J. A. Bioul and G. D. Sutter, "Synthesis of Arithmetic Circuits-FPGA, ASIC, and Embedded Systems", A John Wiley & Sons, inc., publication, vol. 1, **(2006)**.

Authors

Addanki Purna Ramesh has more than 14 years of experience in teaching. He is presently working as Associate professor of Electronics and Communication Engineering at Sri Vasavi Engineering College, Tadepalligudem. He is Life Member of MIETE, Associate Member in Institute of Engineers (India).



A. V. N Tilak has more than 25 years of teaching and research experience. He obtained his Master's degree from Indian Institute of Technology, Kanpur and Ph.D. from Indian Institute of Technology, Madras during 1984 and 1997 respectively. He is presently working as a professor of Electronics and Communication Engineering at V.R.Siddhartha Engineering College, Vijayawada. He is Member of IEEE, Fellow of Institution of Electronics and Communication Engineers (IETE), Fellow of Institute of Engineers (India). He is also life member of Indian Society for Technical Education (ISTE).

Dr. A. Mallikarjuna Prasad has more than 22 years of experience in teaching. He is presently working as a professor of Electronics and Communication Engineering at JNTUK, Kakinada. He is Life Member of ISTE, IETE, ISI, and Society of EMC. He won best teacher award by student evaluation of 2008 batch outgoing students. He has guided about 40 students in M.Tech Instrumentation Engineering and presently guiding 8 research students for their PhD works. His areas of interest are Antennas and Process control Instrumentation. He has 25 publications in various International and National Journals and conferences. He has conducted a "National Workshop on Electromagnetic field applications" in the year 2004.