Mode Enabled Coprocessor for Precision Multipliers

Dinesh Kumar
Research Scholar, SVU,
Gajraula, Amroha (U.P.)

Girish Chander Lall
Former Professor, ECE Deptt.
HCTM Kaithal, Haryana

ABSTRACT

Multiplication and division are the two elementary operations essential for the core computing process or for the arithmetic operation. These two operations are also the most critical functions carried out by the processors, as the multiplication requires more number of steps for the computation, limiting the overall performance of the system, and the division has the highest latency among all arithmetic operations. Thus, high performance multiplication and division algorithms/ architectures, if available, will considerably improve the speeds of processing system. Consequently, the need for faster processing of arithmetic operations, is continuously driving major improvements in processor technologies, as well as the search for new arithmetic algorithms. In the present paper alternate design for single and double precision multiplier processor is presented.

Keywords : Processor, FPGA, Floating Point

1. INTRODUCTION

Solutions for some computational problems are mainly driven by the demanded speed and the budget available. If the highest possible speed is asked for and the algorithm is known, often the only possible solution is to design an expensive application specific integrated circuits (ASIC). The ASIC will certainly fulfill the speed criteria, but if a slight change in the algorithm occurs, the same ASIC cannot be used. These have to be a complicated and expensive redesign. On the other hand, when the required speed is not critical, a slower and cheaper variant is possible. A general purpose processor (GPP) will be fast enough to master the task. The advantage of this solution is that a change in the algorithm can easily be overcome with a change in the software and therefore the GPP can be used again. It is clearly that the GPP has more flexibility than the ASIC, but is slower. Other list of typical applications includes: random logic, integrating multiple SPLDs, device controllers, communication encoding and filtering, small to medium sized systems with SRAM blocks, and many more.

1.1 Reconfigurable FPGAs

Compared with ASICs and with standard microprocessors, FPGAs can be reprogrammed or reconfigured. **Partial reconfiguration** is the process of configuring a portion of a field programmable gate array while the other part is still running / operating. Partial reconfiguration is a design process that allows a limited, predefined portion of an FPGA to be reconfigured while the remaining of the device continues to operate. This is especially valuable where devices operate in a mission-critical environment and cannot be disrupted while subsystems are redefined. The ability to partially reconfigure a device takes the already powerful benefits of reprogram ability to a much higher level. Normally, reconfiguring an FPGA requires it to be held in reset while an external controller reloads a design onto it. Partial reconfiguration allows for critical parts of the design to continue operating while a controller either on the FPGA or off of it loads a partial design into a reconfigurable module. Partial reconfiguration also can be used to save space for multiple designs by only storing the partial designs that change between designs. Partial reconfiguration is not supported on all FPGAs. Xilinx supports partial reconfiguration on Virtex II, Virtex II Pro, and Virtex 4 FPGA lines. From the functionality of the design, partial reconfiguration can be divided into two groups: **Dynamic partial reconfiguration**, also known as an active partial reconfiguration - permits to change the part of the device while the rest of an FPGA is still running. **Static partial reconfiguration** - the device is not active during the reconfiguration process. While the partial data is sent into the FPGA, the rest of the device is stopped (in the shutdown mode) and brought up after the configuration is completed. Organization of paper is : Section 1 describes briefly about multipliers & processor. In Section II brief literature review is presented. Chapter III and IV describe single & double precision multiplier simulations. In the last chapter V discusses conclusion of paper.

1.2 Literature Review

Cui et.al.[1] presented a GaAs floating point single precision multiplier. A modified carry save array is used in conjunction with Booth's algorithm to reduce the partial product addition and interconnection. A special rounding technique called Trailing-1's Predictor is used to speed up the final addition and rounding. Suthikshn Kumar et.al [2] described FPGA implementation of artificial neural networks calls for multipliers of various word length. In their paper, a new algorithm for generating variable word length multipliers for FPGA implementation was presented. Akkas et.al [3] explained that double precision floating-point arithmetic is inadequate for many scientific computations. There paper presented the design of a quadruple precision floating-point multiplier that also supports two parallel double precision multiplications. Marcus G. et.al [4] presented an adder/substractor and a multiplier for single precision floating point numbers in IEEE-754 format. They are fully synthesizable hardware descriptions in VHDL that are available for general and educational use. Each one is presented in a single cycle and pipelined implementation, suitable for high speed computing, with performance comparable to other available implementations.Thapliyal et.al [5] designed a N X N bit parallel overlay multiplier architecture for high speed DSP operations. The architecture is based on the vertical and crosswise algorithm of ancient Indian Vedic Mathematics. Akhter [6] explained a novel technique for digital multiplication that is quite different from the conventional method of multiplication like add and shift. This also gives chances for modular design where smaller block can be used to design the bigger one.

1.3 Single Precision Multiplier

A floating point unit is a part of a computer system specially designed to carry out operations on floating point numbers. Typical operations are addition, subtraction, multiplication, division, and square root. Some systems can also perform various transcendental functions such as exponential or trigonometric calculations, though in most modern processors these are done with software library routines. A FPU, also known as a math coprocessor or numeric coprocessor, is a specialized coprocessor that manipulates numbers more quickly than the basic microprocessor circuitry. The FPU does this by means of instructions that focus entirely on large mathematical operations.

1.3.1 Multiplication Algorithm

Binary floating-point numbers are stored in a sign-magnitude form where the most significant bit is the sign bit, exponent is the biased exponent, and "fraction" or “mantissa” is the significand minus the most significant bit. Given that e_A, e_B and $frac_A$, $frac_B$ are the exponents and significands of the numbers, respectively. A detailed description of the algorithm Urdhva-tiryabhyam method [7] follows:

1. The hidden bit (24th bit) is made explicit. If ^{e}a or $^{e}b = 0$, it is made ‘0’, otherwise a ‘1’. At this point 33 bits are needed to store the number, 8 for the exponent, 24 for the significand and 1 for the sign.

2. The result of the multiplication is given by the formula:
$Sign = sign_A$ xor $sign_B$, $e = e_A + e_B$,
$Frac = frac_A \times frac_B$
The addition of the exponents is a trivial operation as long as we keep in mind that they are biased. This means that in order to get the right result, we have to subtract 127 (bias) from their sum. The sign of the result is just the XOR of the two sign bits. The multiplication of the significands is just an unsigned, integer multiplication.
3. The product of two 24-bit numbers can be 48-bit wide. But only, 24 bits can be accommodated for the significand. Therefore, 48-bit result is rounded up to 24 bits. There are four methods for rounding: Round-to-nearest-even, round-up, round-down and round-to-zero; in which round-to-nearest-even is the most widely used (mode enabled).
Since the result precision is not infinite, sometimes rounding is necessary. To increase the precision of the result and to enable round-to-nearest-even rounding mode, three bits were added internally and temporally to the actual fraction: guard, round, and sticky bit. While guard and round bits are normal storage holders, the sticky bit is turned ‘1’ whenever a ‘1’ is shifted out of range.
4. There must be a leading ‘1’ in the significand of any floating-point number (unless it is not denormalized). To make the MSB ‘1’ in the result, the bits are shifted left, and with each shift, the exponent is incremented by 1. This way normalization is done.
5. The result is assembled into the 32 bit format, neglecting the 24th bit of the significand.

1.3.2 Design And Simulation

The code is written in HDL, synthesized and simulated using Virtex 4 (Device: **xc4vsx35-10-ff668**) and speed grade of -12. RTL schematic is shown in Fig.1.3.1 and Technology schematic is shown in Fig.1.3.2

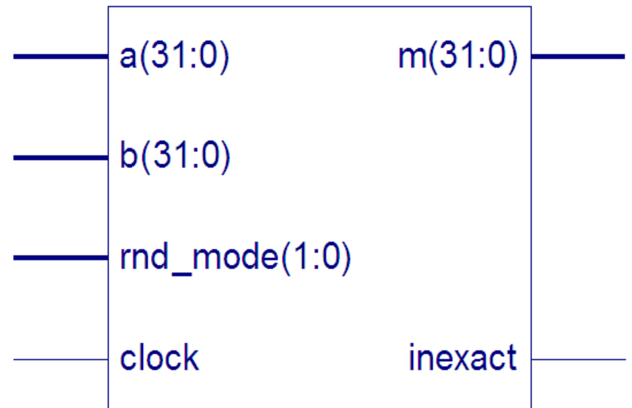

Fig.1.3.1: RTL schematics of FP Multiplier

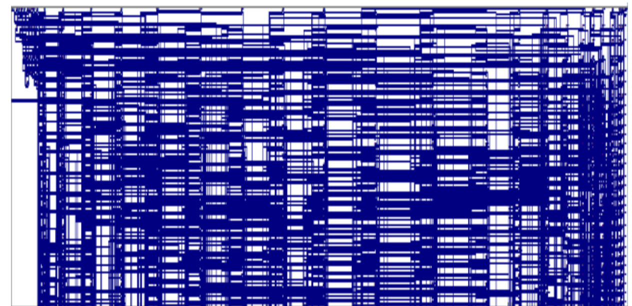

Fig.1.3.2: Technology Schematics FP Multiplier

1.3.3 Results and Discussions

Synthesis results shows 2688 slices are used out of 15360, 5148 numbers of 4 input LUTs are used out of 30720 and number of bonded IOBs are 100 out of 448. This is shown in Fig.1.3.3. **Rounding techniques** are used which make processor mode enabled.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	2688	15360	17%
Number of Slice Flip Flops	269	30720	0%
Number of 4 input LUTs	5148	30720	16%
Number of bonded IOBs	100	448	22%
Number of GCLKs	1	32	3%

Fig.1.3.3: Synthesis results of FP Multiplier

Timing report summary indicates total time taken for process is 50.239 ns out of which 16.136 ns are used for logic and 34.103 ns are utilized for routing. Simulation Results are shown in Fig.1.3.4.

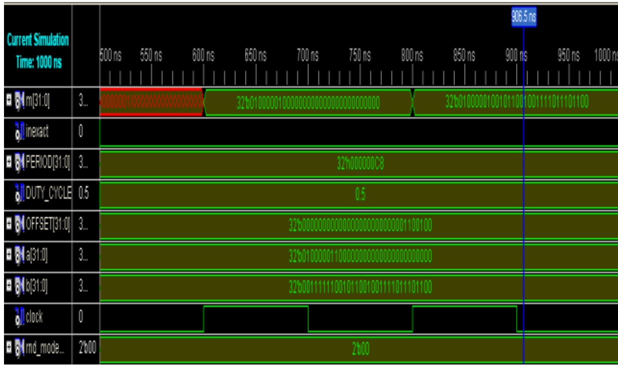

Fig.1.3.4: Simulation results of FP Multiplier

Power analysis shows total power of 0.412W out of which 0.393W is quiescent power and 0.018W dynamic powers. This is shown in Fig.1.3.5

Name	Power (W)	Used	Total Available	Utilization (%)
Clocks	0.018	1	...	...
Logic	0.000	4463	20480	21.8
Signals	0.000	4402	...	...
IOs	0.000	100	320	31.3
DCMs	0.000	0	4	0.0
Total Quiescent Power	0.393			
Total Dynamic Power	0.018			
Total Power	0.412			

Fig. 1.3.5: Power analysis

*1.3.4 Layout of 2*2 multiplier*

Layout of 2*2 multiplier is shown in Fig.1.3.6.

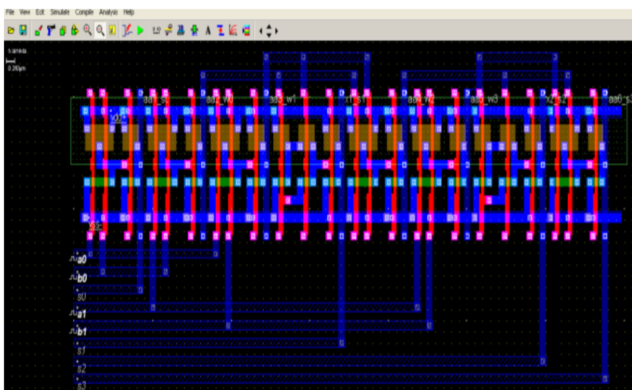

Fig.1.3.6: Layout of 2*2 Multiplier

1.4 Double Precision Multiplier

Design and Simulation

The code is written in HDL, synthesized and simulated using Virtex 4 (Device : **xc4vsx35-10-ff668**) and speed grade of -12. RTL schematics is shown in Fig.1.4.1 and Technology schematics is shown in Fig 1.4.2.

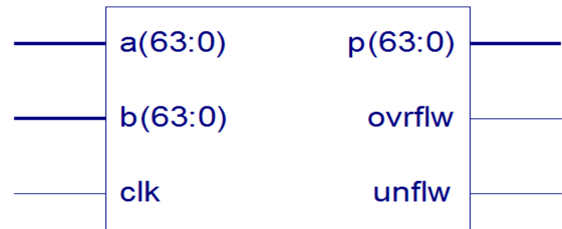

Fig.1.4.1: RTL Schematics of Double Precision Multiplier

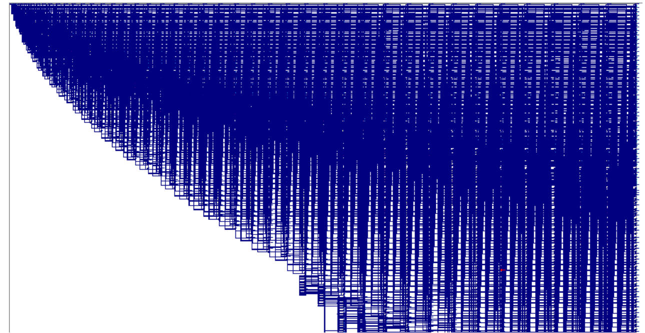

Fig.1.4.2: Technology Schematics of Double Precision Multiplier

Results and Discussions

Synthesis results shows 12447 slices are used out of 15360, 22789 numbers of 4 input LUTs are used out of 30720 and number of bonded IOBs are 195 out of 448. This is shown in Fig1.4.3.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	12447	15360	81%
Number of Slice Flip Flops	144	30720	0%
Number of 4 input LUTs	22789	30720	74%
Number of bonded IOBs	195	448	43%
Number of GCLKs	1	32	3%

Fig.1.4.3: Synthesis results of Double Precision Multiplier

Timing report summary indicates total time taken for process is 203.817 ns out of which 79,879 ns are used for logic and 18.755ns are utilized for routing. Simulation results are shown in Fig.1.44

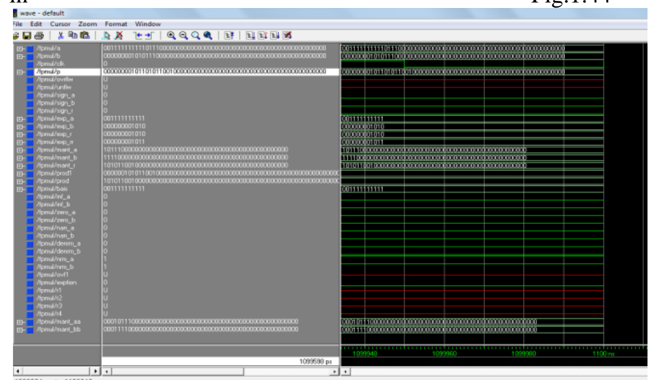

Fig.1.4.4: Simulation Results of Double Precision Multiplier

Comparison with Booth Multiplier

When compared with booth multiplier, it is observed that numbers of slices used are 13055 out of 15360 which is 84% of total available and using Vedic method it is 81 %.which saves 3% area.

1.5 Conclusions

Synthesis results of single precision multiplier shows 2688 slices are used out of 15360, 5148 numbers of 4 input LUTs are used out of 30720 and number of bonded IOBs are 100 out of 448. Timing report summary indicates total time taken for process is 50.239 ns out of which 16.136 ns are used for logic and 34.103 ns are utilized for routing and Synthesis results of double precision multiplier shows 12447 slices are used out of 15360, 22789 numbers of 4 input LUTs are used out of 30720 and number of bonded IOBs are 195 out of 448. Timing report summary indicates total time taken for process is 203.817 ns out of which 79,879 ns are used for logic and 18.755ns are utilized for routing. This is found to be more area efficient than booth multiplier.

2. REFERENCES

[1] Cui, Burgess, Liebelt, Eshraghiant, "A GaAs IEEE Floating Point Standard Single Precision Multiplier", Proceedings of the 12th IEEE Symposium on Computer Arithmetic, Bath, UK, pp-91-97, 1995.

[2] Suthikshn, Kevin ,Patlaniswami, "A Fast-Multiplier Generator for FPGAs" Proceedings of the 8 th International Conference on VLSI Design, India, pp 53-56, 1995.

[3] Akkas¸Schulte, "A Quadruple Precision and Dual Double Precision Floating-Point Multiplier", Proceedings of the Euromicro Symposium on Digital Systems Design, IEEE Computer Society USA, pp-76-79, 2003.

[4] Marcus, Hinojosa, Avila, Flores, "A Fully Synthesizable Single-Precision, Floating-Point Adder/Substractor and Multiplier in VHDL for General and Educational Use", Proceedings of the 5th IEEE International Conference on Devices Circuits and Systems, Dominican Republic, pp-319-323,2004.

[5] Vishal, Thapliyal, "High Speed Efficient N X N Bit Multiplier Based on Ancient Indian Vedic Mathematics", Proceedings of the International Conference on VLSI, Las Vegas, United States, pp 361-365, 2003.

[6] Akhter, "VHDL Implementation of Fast NxN Multiplier", 18th European Conference, Sevilla, Spain, pp 472-475, 2007.

[7] Jagadguru Swami Sri Bharath, Krsna Tirathji, "Vedic Mathematics or Sixteen Simple Sutras from the Vedas", Motilal Banarsidas, Varanasi, India, 1986.

[8] Holt, Hwang, "Finite Precision Error Analysis of Neural Network Hardware Implementations", IEEE Transaction on Computers, Vol. 42, No. 3, pp. 281-290, 1993.

[9] Ciminiera , Valenzano, "Low Cost Serial Multiplier for High Speed Specialized Processors", IEEE Proceedings, Vol. 135, No.5, pp. 259–265, 1988.

[10] Sriraman, Prabakar, "FPGA Implementation of High Performance Multiplier Using Squarer", International Journal of Advanced Computer Engineering & Architecture Vol. 2, No. 2 pp-121-128, 2012.

[11] Myjak, Frias, "A Medium-Grain Reconfigurable Architecture for DSP: VLSI Design, Benchmark Mapping, and Performance", IEEE Transaction on VLSI Systems, Vol.16, No.1, pp.14-23, 2008.

[12] Pradhan, Panda, "Design and Implementation of Vedic Multiplier", A.M.S.E. Journal, Vol.15, No.2, pp.1-19, 2010.

[13] Pradhan, Panda, Sahu, "Speed Comparison of 16x16 Vedic Multipliers", International Journal of Computer Applications,Vol. 21, No.6, pp-16-19, 2011.

[14] Ozbilen Gok, "A Single/Double Precision Floating-Point Multiplier Design for Multimedia Applications", Journal of Electrical & Electronics Engineering, Vol. 1, pp-827-831, 2009.

[15] Tull, "High-Speed Complex Number Multiplier and Inner-Product Processor", IEEE Transactions on Circuits and Systems, Vol. 3, pp.640-643, 2002.

[16] Nan, Chen, "Low-Power Multipliers by Minimizing Switching Activities of Partial Products", IEEE International Symposium on Circuits and Systems, Vol. 4, pp 93-69, 2002.

[17] Chen, Wang, Wu, "Minimization of Switching Activities of Partial Products for Designing Low Power Multipliers," IEEE Transactions on VLSI. Vol. 11, No. 3, pp. 418–433, 2003.