AREA AND POWER PERFORMANCE ANALYSIS OF FLOATING POINT ALU USING PIPELINING

Shubhi Sharma[1], Sarvesh Sharma[2], S.Ravi[3]

[1]*M.Tech VLSI Design, Vellore Institute of Technology, Vellore*
[2]*M.Tech VLSI Design, Vellore Institute of Technology, Vellore*
[3]*Professor, Dept. of electronics Engineering, Vellore Institute of Technology, Tamil Nadu, India*

---***---

Abstract - *We are computing the area, power and timing analysis of floating point arithmetic using pipelining. Nowdays all signal processing algorithms are presented by double precision floating point for hardware implementation as large precision needs large dynamic range. Fixed point has a drawback over floating point, that is fixed point cannot be used for high precision computing as it lacks for large dynamic range. For designing of digital processor computation of arithmetic operations is very important. Which can be easily computed using floating point. In this paper we are computing four unit and one combined unit which are performing the computation 0.0178 times faster at 0.26GHz using Verilog RTL on cadence tool.*

Key Words: Verilog, Floating point, ALU, Fixed point, FPGA.

1.INTRODUCTION

Nowdays all signal handling calculations are exhibited by double precision floating point [2] for hardware implementation as expansive exactness needs extensive element range. Fixed point [1] has a disadvantage over floating point, that is fixed point can't be utilized for high accuracy registering as it needs for extensive element range. At the point when floating point is utilized with FPGA [4] (field programmable Array) it has area and power and timing examination overhead over fixed point. For planning of digital processor calculation of number arithmetic operations is very important. Which can be effectively registered utilizing floating point. Floating point number can be utilized as a part of a few applications like FFT, DSP and where ever high performance is required. We utilize floating point to represent numbers which can't be displayed in whole number because of expansive or little values. At the point when floating point is utilized with FPGA (field programmable array) it has area and power and timing examination overhead over fixed point. Floating point can in like manner be used as a piece of 3D representation which requires parallel execution. Floating point addition and multiplication are two most much of the time utilized operations utilizing double precision. To vary the area, latency and power researchers fused add and multiply unit, add and subtract unit, divide and multiply unit. Using this combination we can have numerous applications. Here we are utilizing Verilog HDL [3] configuration to do pipelined operation and for arithmetic modules. The four operations are addition, subtraction, multiplication, division. Arithmetic logic unit (ALU) [1] is a microprocessor block. All the arithmetic operations happens in this microprocessor block, consequently execution of these block are vital for the entire circuit. Pipelining procedure is utilized to outline computer and digital electronics which use to give us expanded throughput. This paper has execution of double precision that is 64 bits [3] which support four essential operations that is addition, subtraction, multiplication and division [9,10].

2. FLOATING POINT NUMBER

Representation of 64 bit floating point number that is double precision is as shown below:

Double Precision		
Sign	Exponent	Mantissa
63	62.........52	51..................0

Fig -1: 64 bit floating point number

The sign bit is bit number 63rd. "1" means a '–ve' number, and "0" is a '+ve' number. The exponent field is 11 bits in length, involving 62 to 52 bits. The worth in this 11-bit long field is counterbalanced by 1023, so the genuine type used to figure the estimation of the number is 2^(e-1023) . Give us a chance to comprehend it through an illustration, let us take a number 3.5. Presently we will introduce it in floating point group The 63rd bit sign bit is 0 which speak to a '+ve' number. So the exponent become 1024. This can be ascertained by separating 3.5 as (1.75)*2^(1). The example counterbalance is 1023, so you add '1023 + 1' to ascertain the worth for the type field. In this way, bits 62 to 52 will be "1000000000". The main "1" in the type is appeared however is excluded in the arrangement of 64-bit. Most astounding piece of type is 51 which compares to 2^(-1). Bit 50 relates to 2^(-2), and this proceeds down to Bit 0 which compares to 2^(-52). To speak to .75, bits 51 and 50 are 1's, and whatever is left of the bits are zeros. So 3.5 as a double precision floating point number may be:

Sign	Exponent	Mantissa
63	62......52	51...0
0	10000000 000	1100000000000000000000000000000000000000

Fig -2: Double Precision example

2.1 Pipelined adder

We have five stages in an ordinary floating point calculation. They are example exponent difference pre arrangement, addition, standardization and adjusting separately. Give us a chance to take a case of floating point number Z1 =(a1 , e1, f1) and Z2 =(a2, e2, f2). Presently to process for Z1+Z2 we have exponent difference is the initial step d =e1-e2 and if e1<e2 position of type is swapped and the bigger type is given in the outcome. Step 2 incorporates moving of littler portion by d bits to one side and the fraction is realigned now include part.

In the event that the outcome is leaded by zero result will be moved left and type is decremented by driving zero

Right move is performed if result floods and exponent is expanded by 1-bit. This procedure we called as standardization. Round the fraction part come about, if adjusting makes flood than augmentation example by 1-bit by moving right. Alignment stage requires a right shifter that is double the quantity of fraction bits in light of the fact that the bits moved out must be kept up to create the gatekeeper, round and sticky bits required for adjusting. The shifter only needs to shift right by up to 24 places for single-precision or 53 places for double-precision. The normalization stage requires a left shifter equal to the number of exponent bits plus 1 i.e., 25-bits for single-precision and 54-bit for double precision.

Pipelined adder is utilized to build the throughput of the adder unit for this we work all sign, exponential, and division bit independently than type are moved appropriately to liken the type and operation is done on partial piece.

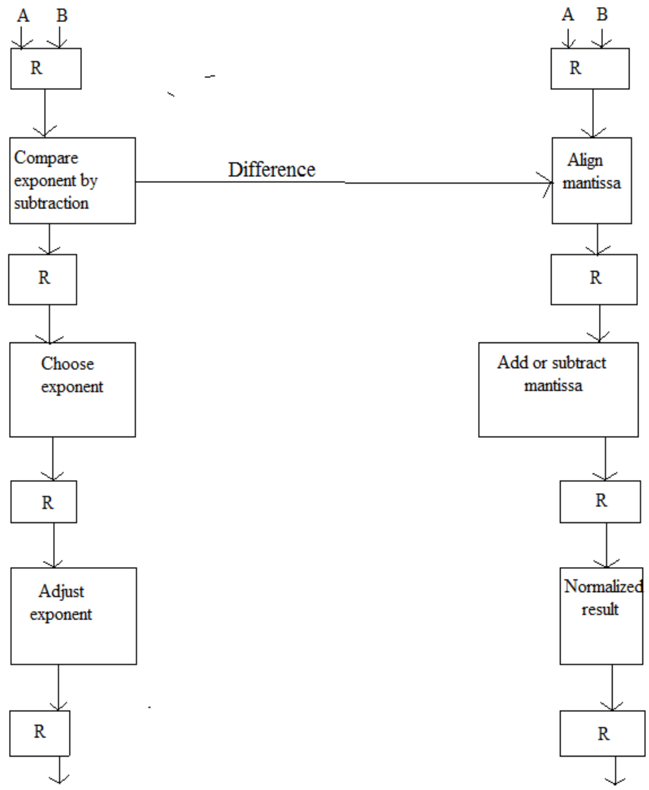

Fig 3: Pipelined adder/subtractor

Table -1: Adder Results

Parameters	Without pipelining	With pipelining	
		45nm	32nm
Power(mW)	0.096	6.35	1.6
Area (μm²)	1195	6351	5969
Timing (ps)	1785	1650	1500

2.2 Pipelined subtractor

We have five stages in an ordinary floating point calculation. They are example exponent difference pre arrangement, addition, standardization and adjusting separately. Give us a chance to take a case of floating point number Z1 =(a1 , e1, f1) and Z2 =(a2, e2, f2). Presently to process for Z1-Z2 we have exponent difference is the initial step d =e1-e2 and if e1<e2 position of type is swapped and the bigger type is given in the outcome. Step 2 incorporates moving of littler portion by d bits to one side and the fraction is realigned now include part. In the event that the outcome is leaded by zero result will be moved left and type is decremented by driving zero Right move is performed if result floods and exponent is expanded by 1-bit. This procedure we called as standardization. Round the fraction part come about, if adjusting makes flood than augmentation example by 1-bit by moving right. Alignment stage requires a right shifter that is double the quantity of fraction bits (i.e., 48- bits for single-accuracy, 106-bits for twofold exactness) in light of the fact that the bits moved out must be kept up to create the gatekeeper, round and sticky bits required for adjusting. The shifter only needs to shift right by up to 24 places for single-precision or 53 places for double-precision. The normalization stage requires a left shifter equal to the number of exponent bits plus 1 i.e., 25-bits for single-precision and 54-bit for double precision.

Pipelined subtractor is utilized to build the throughput of the adder unit for this we work all sign, exponential, and division bit independently than type are moved appropriately to liken the type and operation is done on partial piece.

Table -2: Subtractor Results

Parameters	Without pipelining	With pipelining	
		45nm	32nm
Power(mW)	8.2	5.4	2.35
Area (μm²)	1149	8294	6520
Timing (ps)	1649	2398	2100

2.3 Pipelined multiplication

Give us a chance to have two operands in division they are operand A and operand B with a leading 1 which connotes the standardized number is put away in 53-bit register A and 53-bit register B. Presently we will get 106-piece item on increasing 53-bit An and 53-bit B register esteem. Presently the amalgamation apparatuses Xilinx [1]and Altera does not give us duplication of 53-bit by 53-bit so keeping in mind the end goal to streamline our work we will soften 53-bit up 24-bit and 17-bit littler various units and then at long last their expansion is done at the completion took after by adjusting. At long last enlist will store the 106 piece of result and yield is lessened by making a movement if there is not 1 present in the MSB. The type exponent of operands A and B are included and after that the worth (1022) is subtracted from the aggregate of A and B. In the event that the resultant type is under 0, than the (item) enroll should be right moved by the sum. This worth is put away in storage. The last type of the yield operand become ‘0’ for this situation, and the outcome will be a de-normalized number. In the event that exponent under is more prominent than 52, than the fraction will be moved out of the item enroll, and the yield will be ‘0’, and the "underflow" sign will be stated. The exponent yield from the (fpu _ mul) module is in 56-bit long length register. The MSB is a main "0" to take into overflow in the adjusting module. The primary bit "0" is trailed by the main "1" for standardized numbers, or "0" for de-normalized numbers. At that point the 52 bit long of the fraction take after. 2 additional bits take after the fraction, and are utilized for adjusting purposes. The main additional piece is taken from the following piece after the fraction in the 106 - piece item consequence of the duplicate. The 2nd additional piece is an OR operation of the 52 LSB's of the 106 piece item. Keeping in mind the end goal to increase the throughput as far as range or power or timing in this paper we connected the pipelining concept, pipelining have the idea of dividing the information in example and portion in two subunits and performing them separately, example includes a subpipe than standardization is performed in a typical standardization subpipe to bring the yield.

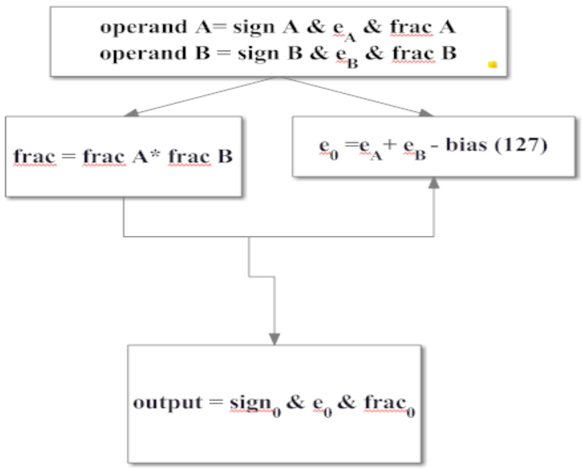

Fig 4: Pipelined Multiplier

Table -3: Multiplication Results

Parameters	Without pipelining	With pipelining	
		45nm	32nm
Power(mW)	5.8	5.6	5.2
Area (μm²)	9482	35719	29781
Timing (ps)	1804	3300	2900

2.4 pipelined divider

The divider gets two 64-bit floating point numbers. To start with these numbers are unloaded by isolating the numbers into sign, example, and mantissa bits. Sign bit of two number perform EXCLUSIVE-OR [2] operation. The exponent of the two numbers are subtracted and after that included with a predisposition number. Mantissa division square performs division utilizing digit repeat calculation. It take minimum 55 clock cycle. After this the yield of mantissa division is standardized, if the MSB of the outcome got is not 1, then it is left moved to make the MSB 1. On the off chance that progressions are rolled out by moving then comparing improvements must be made in type moreover. After mantissa division the yield is 55 bit long. Be that as it may, we require just 53 bit mantissa. So after standardization the 55 bit yield is gone on to the adjusting control. Here adjusting choice is made in view of the mode chose by the user. This mode chooses whether adjusting must be performed – round to closest (code = '00'), round to '0' (code = '01'), round to '+ve' infinity (code = '10'), and round to '-ve' vastness (code = '11'). In view of the adjusting changes into the mantissa comparing changes must be made in the type part also. For round to nearest mode , if the 1st additional leftover portion bit is 1, and the LSB of the mantissa is a 1, then this will become adjusting. For round to 0th mode, no adjusting is operated, unless the yield is sure or '-ve' infinity. This is because of how every operation is executed. For increase and gap, the rest of left the mantissa, thus fundamentally, the operation is as of now adjusting to zero even before the consequence of the operation is gone to the adjusting module. For round to positive infinity mode, the two additional leftover portion bits are checked, and if there is a "1" in either bit, or the sign piece is '0', then the adjusting sum will be activated. Similarly, for round to negative infinity mode, the two additional leftover portion bits are checked, and if there is a "1" in both bits, and the sign piece is '1', then the adjusting sum will be activated. Standardized mantissa will be checked for any exemptions, where the greater part of the extraordinary cases are checked. The extraordinary cases are: Divide by 0 - result is infinity, positive or negative, contingent upon the indication of operand Divide 0 by 0 - result is SNaN [7], and the invalid sign will be attested Divide infinity by infinity result is SNaN, and the invalid sign will be affirmed. Divide by infinity - result is 0, positive or negative, contingent upon the indication of operand and the undercurrent sign will be asserted. Divide overflow result is endlessness, and the flood sign will be stated. Divide underflow result is 0, and the underflow signal will be stated. One or both inputs are QNaN [7] output is QNaN one or both inputs are SNaN yield is QNaN, and the invalid sign will be stated. On the off chance that any of the above cases happens, the special case sign will be stated. On the off chance that the yield is positive vastness, and the adjusting mode is round to zero or round to negative limitlessness, then the yield will be adjusted down to the biggest positive number (exponent = 2046 and mantissa is every one of the 1's). In like manner, if the yield is negative limitlessness, and the adjusting mode is round to zero or round to positive vastness, then the yield will be adjusted down to the biggest negative number. The adjusting of vastness happens in the special cases module, not in the adjusting module. QNaN [7] is characterized as Quiet Not a Number. SNaN [7] is characterized as Signaling Not a Number. In the event that either information is a SNaN, then the operation is invalid. The yield all things considered will be a QNaN. For all other invalid operations, the yield will be a SNaN. In the event that either information is a QNaN, the operation won't be performed, and the yield will be a QNaN. On the off chance that both inputs are QNaNs, the yield will be the QNaN in operand A. The utilization of Not a Number is predictable with the IEEE 754 standard [3]. At last every one of the yields from the sign, type and mantissa are connected to create the last remainder. The entire operation takes around 62 clock cycles. For expanding the recurrence or throughput of the circuit the division step is unrolled and at that point a few pipelining stages are embedded in the middle of every minor operation. The range of a pipeline configuration can be communicated as A Pipe = nc + [n/m]r where c is the combinational territory of a solitary cycle, r is the quantity of bit registers required for a solitary pipeline stage, d is the execution deferral of a solitary iteration, and n is the number of cycles in the successive outline.

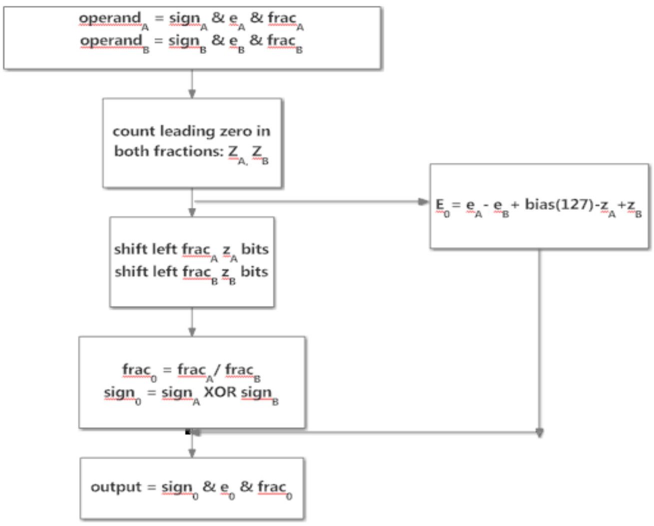

Fig 5: Pipelined dvider

Table -4: divider Results

Parameters	Without pipelining	With pipelining	
		45nm	32nm
Power (mW)	2.6	8.035	3.5
Area (µm²)	10343	12730	11474
Timing (ps)	3621	2798	2298

3. TOP MODULE

The top level, fpu _ double, begins a counter the 1st clock cycle after empower become high. The counter tallies up to the quantity of clock cycle required for this particular operation that will be performed. For expansion, it tallies to 20, for subtraction 21, for duplication 24, and for division 71. When count_ready achieves the predetermined last check, the prepared sign goes high, and the yield will be substantial for the operation being performed. fpu_double contains the instantiations of the other 6 modules, which are 6 separate source records of the 4 operations (include, subtract, duplicate, gap) and the adjusting module and exemptions module. On the off chance that the fpu operation is expansion, and one operand is certain and the other is negative, the fpu_double module will course the operation to the subtraction module. Moreover, if the operation called for is subtraction, and the A operand is certain and the B operand is negative, or if the A operand is negative furthermore, the B operand is certain, the fpu topmodule will course the operation to the expansion module. The sign will likewise be conformed to the right esteem contingent upon the particular case.

Table -4: divider Results

Parameters	With pipelining	
	45nm	32nm
Power (mW)	30.34	16.5
Area (µm²)	72504	20974
Timing (ps)	3800	3200

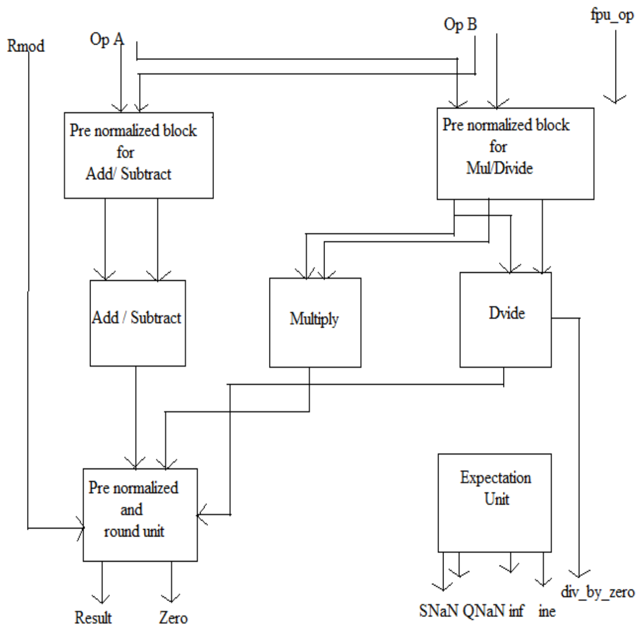

Fig 6: Top module

4. CONCLUSIONS

In this paper different number arithmetic modules are actualized and after that different relative investigations are finished. At last these individual squares are clubbed to make Floating point based ALU in a pipelined way to minimize the power and to expand the working recurrence in the meantime. These relative examinations are done on Altera and Xilinx [1] both. Recreation results are confirmed hypothetically. Verilog HDL) is utilized to plan the entirety ALU piece. In existing configuration, total power is 30.34 mW in 45nm technology and 16.5 in 32nm technology that is 0.0178 times less when appeared differently in relation to the proposed arrangement the working recurrence is 0.26GHz and 0.31GHz.

ACKNOWLEDGEMENT

We acknowledge the help of Prof. S Ravi who guided us helped a lot to get idea and methodologies to do this project. We are also thanking our program chair Prof. Harish Kittur .We are indebted to our lab assistant Prof. Karthikeyan and all other faculties of VLSI System department.

REFERENCES

[1] Rajit Ram Singh, Asish Tiwari, Vinay Kumar Singh, Geetam S Tomar, "VHDL environment for floating point Arithmetic Logic Unit ALU design and simulation", International Conference on Communication Systems and Network Technologies, 2011

[2] H. Yamada, T. Hottat, T. Nishiyama, F. Murabayashi, T. Yamauchi, and H. Sawamoto, "A 13.3ns Double-precision Floating-point ALU and Multiplier", IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1995

[3] Manisha Sangwan, A Anita Angeline, "design and implementation of single precision pipelined floating point co-processor", International Conference on Advanced Electronic Systems (ICAES), 2013

[4] Getao Liang, JunKyu Lee, Gregory D. Peterson, "ALU Architecture with Dynamic Precision Support", Symposium on Application Accelerators in High Performance Computing, 2012

[5] Rathindra Nath Giri,M.K.Pandit, "Pipelined Floating-Point Arithmetic Unit (FPU) for Advanced Computing Systemsusing FPGA", International Journal of Engineering and Advanced Technology (IJEAT), 2012

[6] Geetanjali Wasson, "IEEE-754 compliant Algorithms for Fast Multiplication of Double Precision Floating Point Numbers", International Journal of Research in Computer Science, 2011

[7] Prof. W. Kahan, "IEEE Standard 754 for Binary Floating-Point Arithmetic", Lecture Notes on the Status of IEEE 754, 1997

[8] Mamu Bin Ibne Reaz, Md. Shabiul Islam, Mohd. **S.** Sulaiman, "Pipeline Floating Point ALU Design using VHDL", IEEE International Conference on Semiconductor Electronics ICSE, 2002

[9] Kui YI, Yue-Hua DING, "32 bit Multiplication and Division ALU Design Based on RISC Structure", International Joint Conference on Artificial Intelligence, 2009

[10] Florent de Dinechin, Hong Diep Nguyen, Bogdan Pasca, "Pipelined FPGA Adders", International Conference on Field Programmable Logic and Applications, 2010

[11] Alok Baluni, Farhad Merchant, S.K. Nandy, S. Balakrishnan, "A Fully Pipelined Modular Multiple Precision Floating point Multiplier With Vector Support", International Symposium on Electronic System Design, 2011

[12] Prashanth B.u.v P.Anil Kuma, .G Sreenivasulu, "Design & Implementation of Floating point ALU on a FPGA

Processor", International Conference on Computing, Electronics and Electrical Technologies [ICCEET], 2012

[13] Jinde Vijay Kumar, Boya Nagaraju, Chinthakunta Swapna, Thogata Ramanjappa, "Design and Development of FPGA Based Low Power Pipelined 64-Bit RISe Processor with Double Precision Floating Point Unit", International Conference on Communication and Signal Processing, 2014

[14] Jean Pierre David, "Low Latency Solver for Linear Equation Systems in Floating Point Arithmetic", International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2015

[15] Romesh M. Jessani and Michael Putrino, "Comparison of Single- and Dual-Pass Multiply-Add Fused Floating-Point Units", IEEE transactions on computers, 1998

[16] Zhaolin Li, Xinyue Zhang, Gongqiong Li, Runde Zhou, "Design of A Fully Pipelined Single-Precision Floating-Point Unit", International Conference on ASIC, 2007