Low Power VLSI Architecture for FP Adder With VHDL in DSP Application

Ali Farmani
Lorestan University, Electrical And Computer Engineering Department
Ali_farmani88@ms.tabrizu.ac.ir

Abstract: *In this paper, we present the design of a low power floating-point adder for DSP and FPGA application. We provide synthesis results shown the estimated power consumption for our design when it is pipelined and glitching and re-timing and clock gating. Our work is an important design supply for development of this unit design on DSP. All components within the FP adder and known algorithm are researched and design to provide elasticity to designers as an alternative to brilliant property where they have no control over the design. Each of the operation is researched for different design and then result onto a Altera FPGA device to be chosen for power efficient. Our design of the basic algorithm occupied 370 slices and had an overall delay of 31 ns. The basic algorithm was pipelined into five stages to run at 100 MHz which took an area of 324 slices and power is 30mw*[1]**.**

KEYWORDS: *Digital signal processing, Floating Point Adder, FPGA, VHDL, Pipeline, Glitching, Micro-Architecture, Re-timing, Clock gating, Operand Isolation*.

1. Introduction

High performance floating point adders are essential building blocks of microprocessors and floating point DSP data paths. Since the hardware implementation of floating point addition (FP addition) involves the realization of a multitude of distinct data processing sub-units that endure a series of power consuming transitions during the course of their operations [1–4], the power consumption of floating point adders are, in general quite significant in comparison to that of their integer counterparts.Owing to the presence of a relatively high traffic of floating point additions in microprocessors[5–7] and DSPs. In digital CMOS implementations, the power consumption and speed performance of functional units are, to a large extent, susceptible to algorithmic design decisions [8]. These decision sinfluence the switching activities, fan outs, layout complexity, logic depths, operand scalability and pipeline ability of functional units. Among the above, switching activities, fan outs and layout complexity are directly related to the power consumption. The higher the values of these parameters, the higher the power consumption. Architectural design for low power operation targets the minimization of these parameters. Traditionally, the architecture of floating point adders had been centered around the sequential machine philosophy. With an ever escalating demand for high performance floating point units, newer design approaches are emerging[5–7,9–11]. With new designs, throughput acceleration is achieved through operational parallelism. In [5,6] and [7] Oberman etal. proposed a novel architecture that incorporates concurrent, speculative computation of three possible results of an FP addition. These results are typified by the time complexities of their operations: the hardware realization of ceratin cases of floating point addition can be simplified so that the execution time of such operations are reduced. With this, the variable latency architecture reported in [5, 6] and [7] produces results within 1, 2 or 3 cycles, by virtue of which the average latency of FP additions is reduced. The fusion of multiply and accumulate operations in floating point multiply-accumulate (MAC) units [9], and the concurrent evaluation of leading zeros with significand addition (leading zero anticipation) [9–11] are other notable examples that exploit parallelism for latency reduction. The operation of rounding is an integral part of floating point addition. While MAC fusion and leading zero anticipatory logic provide throughput acceleration, speculative computing for rounding [12] also enhances the throughput of floating point units. Though the concept of operational parallelism can be put to advantageous use as far as throughput acceleration is concerned, and the trading of speed performance for power reduction is also not infeasible, this approach, however cannot guarantee power reduction in performance critical applications. The main objective of their implementation was to achieve IEEE standard accuracy with reasonable performance parameters. This is claimed to be the first IEEE single precision floating-point adder implementation on a FPGA, before this, implementation with only 18-bit word length was present [2]. Floating-point addition is the most frequent floating-point operation and accounts for almost half of the scientific operation. Therefore, it is a fundamental component of math coprocessor, DSP processors, embedded arithmetic processors, and data processing units. These components demand high numerical stability and accuracy and hence are floating-point based. Floating-point addition is a costly operation in terms of hardware and timing as it needs different types of building blocks with variable latency. A lot of work has been done to improve the overall latency of floating-point adders. Various algorithms and design approaches have been developed by the Very Large Scale Integrated(VLSI) circuit community [3-4,9-12] over the span of last two decades. Binary floating-point arithmetic is usually sufficient forscientific and statistics applications. However, it is not sufficient for many commercial applications and database systems, in which operations often need to mirror manual calculations. Therefore, these applications often use software to perform decimal floating - point arithmetic operations. Although this approach eliminates errors due to converting between binary and decimal numbers and provides decimal rounding to mirror manual calculations, it results in long latencies for numerically intensive commercial applications. Because of the growing importance of decimal floating-point arithmetic, specifications for it have been added to the draft revision of the IEEE-754 Standard for Floating-Point Arithmetic (IEEE P754)[5]. The most important functionality of FPGA devices is their ability to reconfigure when needed according to the design need. In 2003, J.Liang, R.Tessier and O.Mencer[6] developed a tool which gives the user the option to create vast collection of floating-point units with different throughput, latency, and area characteristics. One of the most recent works published related to our work is published by G.Govindu, L.Zhuo, S.Choi, and V.Prasanna[7] on the analysis of high-performance floating-point arithmetic on FPGA.

2. Algorithm of floating point adder

In this section we express design and implementation algorithm of floating point adder with single precision .

A. Fixed Point and Floating Point Representations

Every real number has an integer part and a fraction part; a radix point is used to differentiate between them. The number of binary digits assigned to the integer part may be different to the number of digits assigned to the fractional part. A generic binary representation with decimal conversion is shown in table 1.

Table 1: Binary representation and conversion to decimal of a numeric.

Number	Integer part				Binary point	Fraction part		
Binary	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
decimal	8	4	2	1	.	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$

B. Fixed-Point Representation

A representation, in which a real number is represented by an approximation to some fixed number of places after the radix or decimal point, is called a fixed-point representation. Usually the radix point is placed next to the least significant bit thus only representing the integer part. The main advantage of this kind of representation is that integer arithmetic can be applied to them and they can be stored using small values. This helps making the operations faster and area efficient. The main disadvantage is that a fixed-point number has limited or no flexibility, i.e., number of significant bits to the right of the decimal point. Some of the other disadvantages are that the arithmetic operations based on this representation can go into overflow and underflow often. The fixed-point number also has a limited integer range and it is hard to represent very small and big number in the same representation. These are some of the reasons why floating-point representation and arithmetic was evolved to take care of these disadvantages.

C. 2's Complement Representation

In order to represent both positive and negative fixed-point numbers, 2's complement representation is used. Positive 2's complement numbers are represented as simple binary. Negative number is represented in a way that when it is added to a positive number of same magnitudes the answer is zero. In 2's complement representation, the most significant bit is called the sign bit. If the sign bit is 0, the number is non-negative ,i.e., 0 or greater. If the sign bit is 1, the number is negative or less than 0. In order to calculate a 2's complement or a negative of a certain binary integer number, first 1's complement, i.e., bit inversion is done and then a 1 is added to the result.

D. Floating-Point Representation

In general, a floating-point number will be represented $\pm d.dd..d \times \beta^E$, where $d.dd..d$ is called the significand and has p digits also called the precision of the number, and β is the base being 10 for decimal, 2 for binary or 16 for hexadecimal. If $\beta = 10$ and $p = 3$, then the number 0.1 is represented as 1.00×10^{-1}. If $\beta = 2$ and $p = 24$, then the decimal number 0.1 cannot be represented exactly, but is approximately $1.10011001100110011001101 \times 2^{-4}$. This shows a number which is exactly represented in one format lies between two floating-point numbers in another format. Thus the most important factor of floating-point representation is the precision or number of bits used to represent the significands. Other important parameters are $E_{\max}$ and $E_{\min}$, the largest and the smallest encoded exponents for a certain representation, giving the range of a number.

E. IEEE Floating Point Representation

The Institute of Electrical and Electronics Engineering (IEEE) issued 754 standard for binary floating-point arithmetic in 1985 [15]. This standardization was needed to eliminate computing industry's arithmetic vagaries. Due to different definitions used by different vendors, machine specific constraints were imposed on programmers and clients. The standard specifies basic and extended floating-point number formats, arithmetic operations, conversions between various number formats, and floating-point exceptions. This section goes over the aspects of the standard used in implementing and evaluating various floating-point adder algorithms.

F. Basic Format

There are two basic formats described in IEEE 754 format, double-precision using 64-bits and single-precision using 32-bits. Table 2 shows the comparison between the important aspects of the two representations.

Table 2: Single and double precision format summary

Format	Precision	Emax	Emin	Exponent width	Format width
Single	24	+127	-126	8	32
Double	53	+1023	-1022	11	64

The single-precision floating-point number is calculated as $(-1)^{s} \times 1.F \times 2^{(E-127)}$The sign bit is either 0 for non-negative number or 1 for negative numbers. The exponent field represents both positive and negative exponents. To do this, a bias is added to the actual exponent. For IEEE single-precision format, this value is 127, for example, a stored value of 200 indicates an exponent of (200-127), or 73. The mantissa or significand is composed of an implicit leading bit and the fraction bits, and represents the precision bits of the number. Exponent values (hexadecimal) of 0xFF and 0x00 are reserved to encode special numbers such as zero, denormalized numbers, infinity, and NaNs. The mapping from an encoding of a single-precision floating-point number to the number's value is summarized in Table 3.

Table 3: IEEE 754 single precision floating-point encoding

Sign	Exponent	Fraction	Value	Description
S	0Xff	0x00000000	$(-1)^{s}\infty$	Infinity
S	0xFF	F≠0	NaN	Not a Number
S	0x00	0x00000000	0	Zero
S	0x00	F≠0	$(-1)^{s} \times 0.F \times 2^{(E-126)}$	DenormalNum
S	0x00< E < 0xFF	F	$(-1)^{s} \times 1.F \times 2^{(E-127)}$	NormalNum

G. Normalized numbers

A floating-point number is said to be normalized if the exponent field contains the real exponent plus the bias other than 0xFF and 0x00. For all the normalized numbers, the first bit just left to the decimal point is considered to be 1 and not encoded in the floating-point representation and thus also called the implicit or the hidden bit. Therefore the single-precision representation only encodes the lower 23 bits.

H. Denormalized numbers

A floating-point number is considered to be denormalized if the exponent field is 0x00 and the fraction field doesn't contain all 0's. The implicit or the hidden bit is always set to 0. Denormalized numbers fill in the gap between zero and the lowest normalized number.

I. Infinity

In single-precision representation, infinity is represented by exponent field of 0xFF and the whole fraction field of 0's.

J. Not a Number (NaN)

In single-precision representation, NaN is represented by exponent field of 0xFF and the fraction field that doesn't include all 0's.

K. Zero

In single-precision representation, zero is represented by exponent field of 0x00 and the whole fraction field of 0's. The sign bit represents -0 and +0, respectively.

L. Rounding Modes

Rounding takes a number regarded as infinitely precise and, if necessary, modifies it to fit in the destination's format while signaling the inexact exception. Thus the rounding mode affects the results of most arithmetic operations, and the thresholds for overflow and underflow exceptions. In IEEE 754 floating point representation, there are four rounding modes defined: round towards nearest even (REN), round towards $-\infty$ (RP), round towards $+\infty$ (RM), and round towards 0 (RZ). The default rounding mode is REN and is mostly used in all the arithmetic implementations in software and hardware. In order to evaluate different adder algorithms, we are also interested in only the default rounding mode i.e. REN. In this mode, the representable value nearest to the infinitely precise result is chosen. If the two nearest representable values are equally near, the one with its least significant bit equal to zero is chosen.

M.Exceptions

The IEEE 754 defines five types of exceptions: overflow, underflow, invalid operation, inexact result, and division-by-zero. Exceptions are signaled by setting a flag or setting a trap. In evaluating hardware implementations of different floating-point adder algorithms, we only implemented overflow and underflow flags in our designs as they are the most frequent exceptions that occur during addition.

N. Standard Floating Point Addition Algorithm

This section will review the standard floating point algorithm architecture, and the hardware modules designed as part of this algorithm, including their function, structure, and use. The standard architecture is the baseline algorithm for floating-point addition in any kind of hardware and software design [16].

3. Algorithm

Let s1; e1; f1 and s2; e2; f2 be the signs, exponents, and significands of two input floating-point operands, N1 and N2, respectively. Given these two numbers, Figure 1 shows the flowchart of the standard floating-point adder algorithm. A description of the algorithm is as follows.

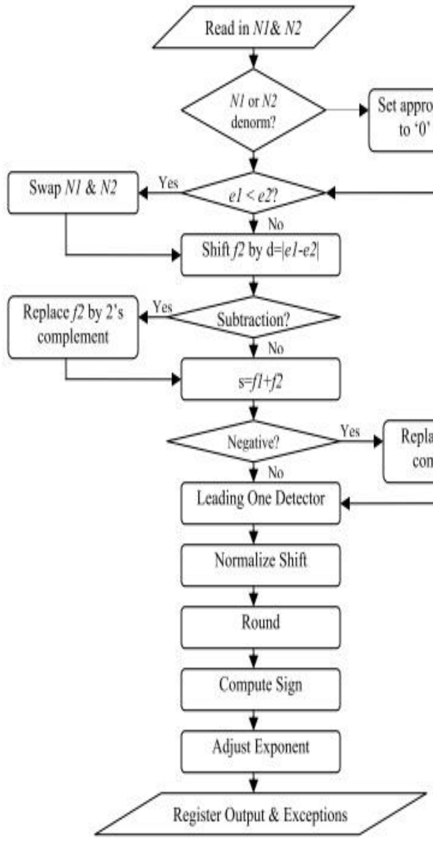

Fig. 1. Flow chart for standard floating-point adder

1. The two operands, N1 and N2 are read in and compared for denormalization and infinity. If numbers are denormalized, set the implicit bit to 0 otherwise it is set to 1. At this point, the fraction part is extended to 24 bits.

2. The two exponents, e1 and e2 are compared using 8-bit subtraction. If e1 is less than e2, N1 and N2 are swapped i.e. previous f2 will now be referred to as f1 and vice versa.

3. The smaller fraction, f2 is shifted right by the absolute difference result of the two exponents' subtraction. Now both the numbers have the same exponent.

4. The two signs are used to see whether the operation is a subtraction or an addition.

5. If the operation is a subtraction, the bits of the f2 are inverted.

6. Now the two fractions are added using a 2's complement adder.

7. If the result sum is a negative number, it has to be inverted and a 1 has to be added to the result.

8. The result is then passed through a leading one detector or leading zero counter. This is the first step in the normalization step.

9. Using the results from the leading one detector, the result is then shifted left to be normalized. In some cases, 1-bit right shift is needed.

10. The result is then rounded towards nearest even, the default rounding mode.

11. If the carry out from the rounding adder is 1, the result is left shifted by one.

12. Using the results from the leading one detector, the exponent is adjusted. The sign is computed and after overflow and underflow check, the result is registered.

A. *Micro-Architecture*

Using the above algorithm, the standard floating point adder was designed. The detailed micro-architecture of the design is shown in Figure 2. It shows the main hardware modules necessary for floating-point addition. The detailed description and functionality of each module will be given later in this chapter.

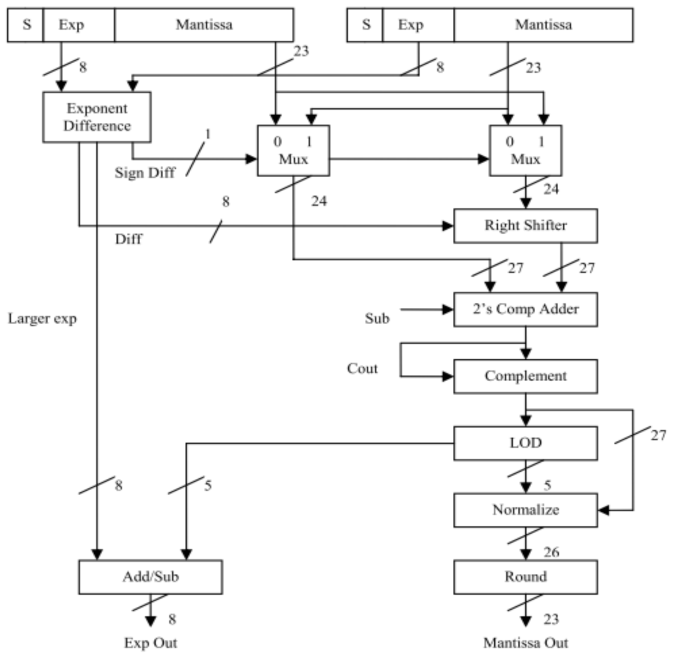

Fig. 2. Micro-architecture of standard floating-point Adder

The main hardware modules for a single-precision floating-point adder are the exponent difference module, right shift shifter, 2's complement adder, leading one detector, left shift shifter, and the rounding module. The bit-width as shown in Figure 3 and following figures is specifically for single-precision addition and will have to be changed for any other format.

B. *2's Complement Adder*

2's complement adder is a simple integer addition process which adds or subtracts the pre-normalized significands. Figure 3 shows the hardware description of a 2's complement adder.

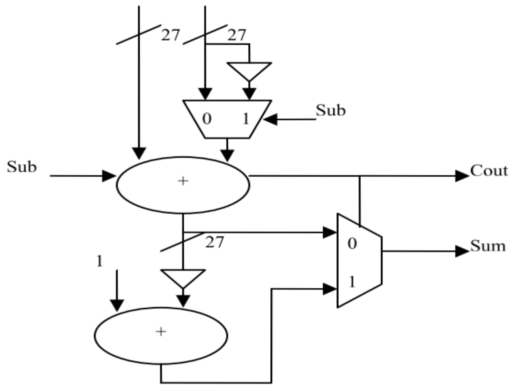

Fig. 3. Hardware implementation for 2's complement adder

The two 27 bit significands enter the module. The signs are multiplexed using an XOR gate to determine if the operation is addition or subtraction. In case of subtraction, the sub bit is 1 otherwise it's 0. This signal is used to invert one of the operands before addition in case of subtraction. A 27-bit adder with sub bit being the carry-in computes the addition. The generated carry out signal determines the sign of the result and is later on used to determine the output sign. If the result is negative, it has to be inverted and a 1 has to be added to the result.

C. *Leading One Detector*

After the addition, the next step is to normalize the result. The first step is to identify the leading or first one in the result. This result is used to shift left the adder result by the number of zeros in front of the leading one. In order to perform this operation, special hardware, called Leading One Detector (LOD) or Leading Zero Counter (LZC), has to be implemented. There are a number of ways of designing a complex and complicated circuit such as LOD. A combinational approach is a complex process because each bit of the result is dependant on all the inputs. This approach leads to large fan-in dependencies and the resulting design is slow and complicated. Another approach is using Boolean minimization and Karnaugh map, but the design is again cumbersome and unorganized. The circuit can also be easily described behaviorally using VHDL and the rest can be left to Xilinx ISE or any synthesis tool. In our floating-point adder design, we used the LOD design which identifies common modules and imposes hierarchy on the design. As compared to other options, this design has low fan-in and fan-out which leads to area and delay efficient design [17] first presented by Oklobdzija in 1994.

D. *Oklobdzija's LOD*

The first step in the design process is to examine two bits case shown in Table 4. The module is named as LOD2. The pattern shows the possible combinations. If the left most bit is 1, the position bit is assigned 0 and the valid bit is assigned 1. The position bit is set to 1 if the second bit is 1 and the first bit is 0. The valid bit is set to 0 if both the bits are 0.

Table 4: Truth table for LOD2

Pattern	Position Bit	Valid Bit
1x	0	1
01	1	1
00	x	0

The logic for LOD2 is straightforward and shown in Figure4.

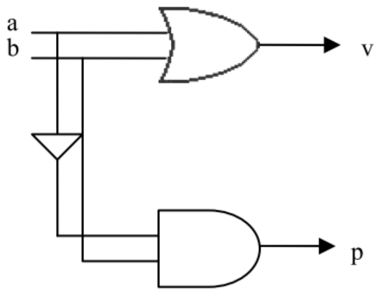

Fig. 4. Hardware implementation for LOD2

The two bit case can be easily extended to four bits. Two bits are needed to represent the leading-one position. The module is named LOD4. The inputs for the LOD4 are the position and valid bits from two LOD2's, respectively. The two level implementation of LOD4 is shown in Figure 5.

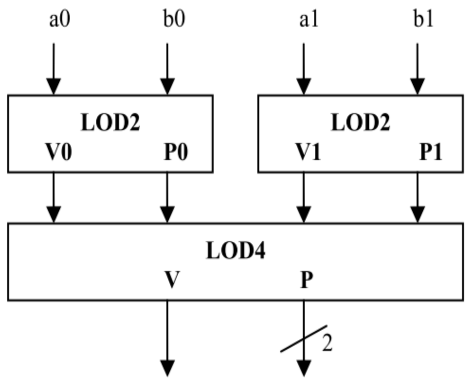

Fig. 5. Two level implementation of 4 bit LOD

The truth table examining the LOD4 is shown in Table 5. The second bit of the LOD4 position bits is selected using the valid bit, V0 of the first LOD2. V0 is inverted to get the first position bit. The output valid bit is the OR of the two input valid bits.

Table 5: Truth table for LOD4 with inputs from two LOD2's

Pattern	P0-LOD2	P1-LOD2	V0-LOD2	V1-LOD2	P-LOD4	V-LOD4
1xxx	0		1		00	1
01xx	1		1		01	1
001x		0	0	1	10	1
0001		1	0	1	11	1
0000	0	0	0	0		0

The hardware implementation of the LOD4 is shown in Figure 6.

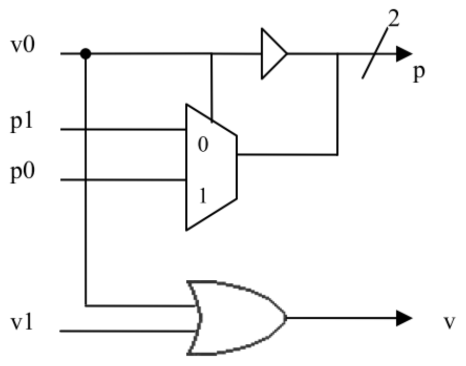

Fig. 6. LOD4 logic implementation

The implementation of the behavioral LOD is done entirely by the Xilinx synthesizer which results in a cumbersome design and adds routing delays. On the other hand, the basic module for implementation described by Oklobdzija is a two to one multiplexer, which are implemented using the inbuilt function generators of the slices in the CLBs of the spartan FPGA. Each connection is defined, thus minimum routing delay is expected, and results in better propagation delay and area compared to behavioral implementation. The behavioral model had a negligibly smaller combinational delay, and smaller area, and is therefore used in our implementation. This result was unexpected because a behavioural implementation has given a better timing and area numbers compared to the VHDL operator which uses inbuilt shift registers in the CLBs. For a single precision floating-point adder the maximum amount of left shift needed is 27. The hardware for the behavioral left shifter is designed to only accommodate the maximum shift amount. As we have no control over the hardware implementation in VHDL shifter, it implements hardware for shift amounts greater than 27, thus resulting in bigger area and delay compared to behavioral shifter. Only in case when the carry out from the adder is 1 and the operation is addition, the result is shifted right by one position.

4. POWER OPTIMIZATION TECHNIQUES

Platform dependent power optimization techniques are implemented by using the opportunites which are provided by the implementation platform. One of the power optimization techniques are sleep mode operation which is called as power gating. The static power consumption of a CMOS circuit is caused by the leakage currents of transistors and pn junctions [20]. Especially SRAM based FPGA platform causes the circuit consumes a huge amount of static power caused by the leakage currents when the circuit is off. Power gating method prevents the power consumption by using sleep mode for the states that the circuit is off [17]. There are memory blocks in the FPGA's which cause the dynamic power consumption. If there are input ports to read or write on these memory blocks, these inputs are allowed to disable for the memory blocks which are not used at that time. In this way, dynamic power consumption of unused memory blocks is prevented. [17]. The clock signal in the FPGA has to reach for every single sequential block; so it has a long routing line. These long routing lines causes power dissipation by charging and discharging the nodes capacitance, which is therefore also referred to as the capacitive power dissipation. It is also obvious that clock signal has a high frequency of logic level change; this is why its dynamic power consumption is high. So there is a way of power optimization by preventing of clock routing to the blocks which are unused. This feature is available on some of FPGA platforms [17].

1. *Platform Independent Power Optimization Techniques*

A. *Glitching*

Glitches are unwanted transitions of a signal after an input change until the final output value is reached. This behavior is due to different arrival times of signals to a gate, called logic hazards. Figure 7 shows the circuit for the logic equation Q = AB + BC which exhibits a static-1 hazard. When the inputs A and C are logic 1 any change on B will cause a transition on Q. There are two paths for B to the output Q where one path contains an inverter. This causes a slightly longer delay, resulting in a glitch in the output Q [20]. More complex circuits e.g. ripple carry adders, amplify this problem. In typical combinational circuits glitching accounts for between 10% and 40% of the dynamic power consumption. Hazards and therefore glitches can be avoided at the cost of more circuitry [21].

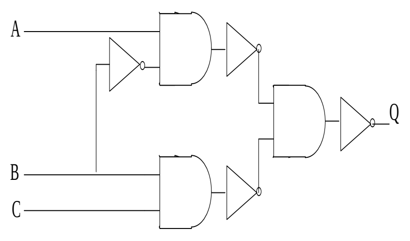

Fig. 7. Glitch caused by hazard.

There are two types of ways in order to solve this glitching problem in the circuit: The first method, which is widely used in our implementation, is to place register blocks between large combinational circuits. These register blocks not only decreases the logic deepness in the circuit, but also increases the clock frequency in the circuit.However, to place these register blocks increases the data processing time, This method is shown in Figure 8.

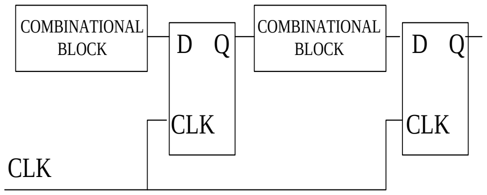

Fig. 8. Reducing glitches by adding register blocks.

Second method is to solve glitch problem by reducing the logic deepness of the circuit. This solution is applied to the circuit during the HDL code implementation by using some coding hints. For example the circuit in Figure 9 can be converted into a circuit as Figure 10 by doing some changes in the HDL where if, elsif and else blocks are stated. In this way, both the logic deepness of the circuit and the amount of the glitches are reduced [22].

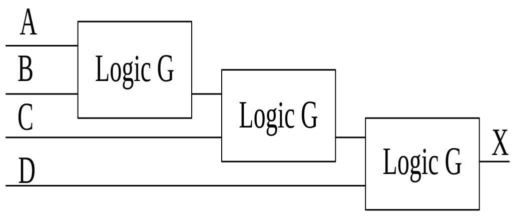

Fig. 9. The circuit that has unbalanced routing delays.

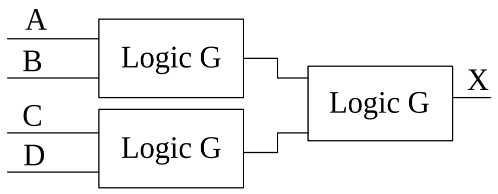

Fig.10. The circuit that has balanced routing delays.

B. *Clock gating*

Figure 11 shows a typical implementation of a synchronous register with enable. We assume that a register is multiple bits wide and consists of one flip-flop per bit. The register is disabled when the enable signal is at logic 0. Its output is fed back to its input through the multiplexer. When the enable signal is at logic 1 the register can load new values from data in. In this design each flip-flop of the register requires a multiplexer at its data input [20].

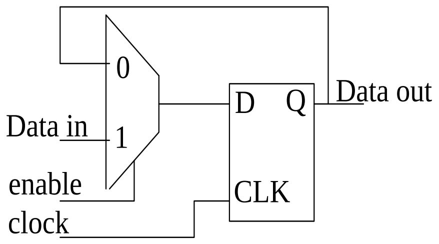

Fig. 11. Enable register with multiplexer.

Furthermore the clock network has to drive each flipflop. Clock gating provides a way to disable the clock signals for a register, therefore eliminating the need for separate multiplexers for each input bit. Figure 12 shows such a design. The enable signal is usually the output of some combinatorial logic and may contain glitches. The latch prevents glitches from the enable signal to propagate to the clock input of the register. The AND gate performs the actual gating. Clock gating replaces the multiplexers with a single clock gating cell and isolates the register clock from the global clock. The clock gating cell, containing a latch and an AND gate, consumes more power than a single bit multiplexer. However, when this technique is applied to multiple bit registers it can conserve both, static and dynamic power. We observed savings even at registers that were only 8-bits wide [20].

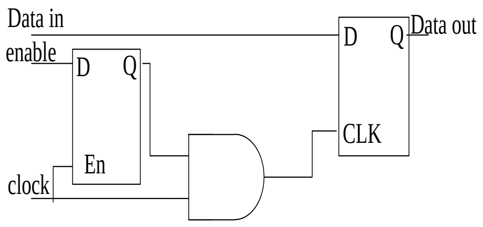

Fig. 12. Clock gated register.

1) Operand Isolation

Operand Isolation is a method to selectively stop data from entering a block of complex combinatorial logic, causing many transitions and therefore dynamic power consumption, when the output is discarded by either an unselected multiplexer or a currently disabled register. Figure 13 shows an example where changes to the input A consume power even when the output A' is not used.

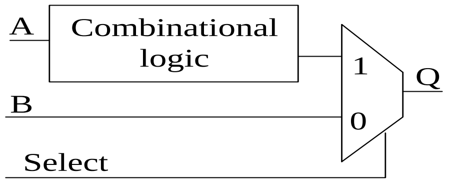

Fig. 13. Design without operand isolation.

To prevent this unnecessary power consumption isolation logic can be added at the input to the complex combinatorial logic. It prevents changes to input A from propagating through the combinatorial logic. The isolation logic usually consists of either AND or OR gates depending on the specific application. The example in Figure 14 uses an AND gate for operand isolation. The combinatorial logic only receives the input A when its output A' is selected by the multiplexer. Otherwise its input is 0. In this way, it is prevented unnecessary power consumption when control signal Select is not logic1, which means the output of combinatorial logic is not used [20].

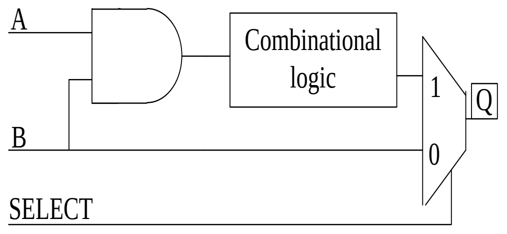

Fig. 14. Design with operand isolation.

2) Re-timing

Retiming for low-power is the process of positioning new or moving existing flip-flops so that they separate parts of the circuit that cause glitching from parts that have high input capacitance. As glitches do not get propagated through flip-flops this technique significantly reduces the switching activity of the high input capacitance part of the circuit and hence reduces the dynamic power consumption [20]. The critical path in Figure 15 is decreased by changing the places of registers. The circuit in Figure 16 is redrawn in Figure 11 after being applied this retiming method [22].

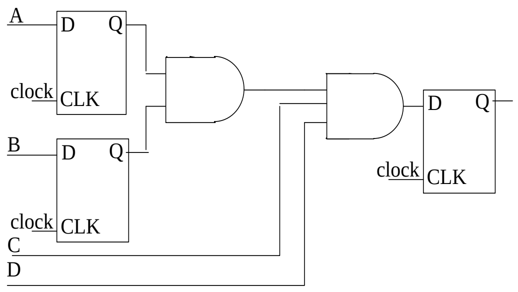

Fig. 15. Design without re-timing.

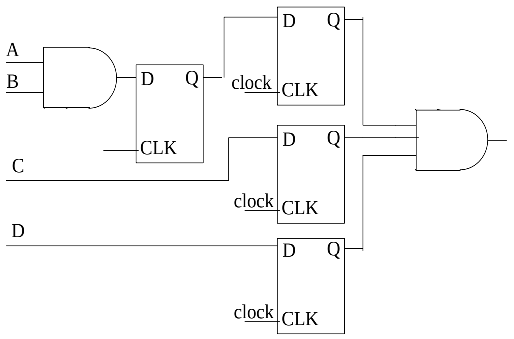

Fig. 16. Design with re-timing.

5. Timing and Area Analysis

The standard single precision floating point adder is synthesized, placed, and routed for spartan FPGA device using Xilinx ISE 12. The minimum clock period reported by the synthesis tool after placing and routing was 31 ns. The levels of logic reported were 44. That means the maximum clock speed that can be achieved for this implementation is 37 MHz. The number of slices reported by the synthesis tool was 370. All this information will be used as a base to analyze improved floating-point adder algorithms.

A. *Five Stage Pipeline Standard Floating Point Adder Implementation*

In order to decrease clock period, to run the operations at a higher clock rate, and to increase speedup by increasing the throughput, pipelining is used. Pipelining is achieved by distributing the hardware into smaller operations, such that the whole operation takes more clock cycles to complete but new inputs can be added with every clock cycle increasing the throughput. Pipelining of floating-point adder has been discussed in a number of previous research papers [9,10]. Minimum, maximum, and optimum number of pipeline stages for a 32 bit floating-point number has been given based on the factor of frequency per area (MHz/Slices). According to these studies, 5 pipeline stages are the optimum for single-precision adder implementation. In order to achieve this, all of the hardware modules have to be sub-pipelined within themselves. In order to analyze effects of pipelining on floating-point adder implementations on FPGAs, we will compare our implementation results with Xilinx IP Core by Digital Core Design [12].

B. *Micro-Architecture*

Figure 17 shows the micro-architecture of five stage pipeline implementation of the standard floating-point adder algorithm implementation. The levels of pipeline chosen are purely based on comparison with the Xilinx IP Core and are entirely a design choice according to the design needs. Five is a good choice because anymore stages will need sub pipelining the modules. The placement of the registers in order to put stages is shown as the dotted line in Figure 17. The main theory behind pipelining is to decrease the clock period thus increasing the overall clock speed that the application can run. Adding pipeline stages exploits the D flip-flops in the slices already being used for other logic and thus doesn't increase the area significantly. Pipelining also helps increase throughput as after the first five clock cycles a result is produced after every clock cycle.

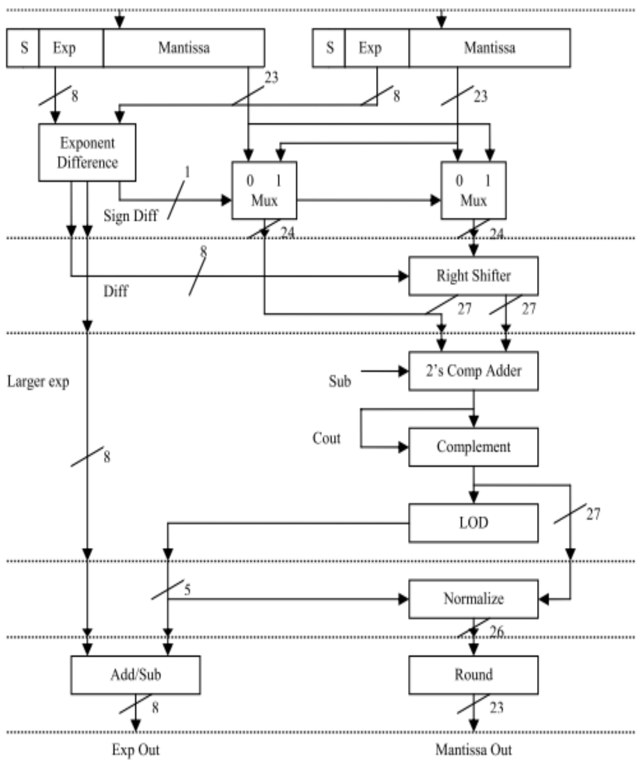

Fig. 17. Micro-architecture of 5 stage pipeline standard floating-point adder

In the first stage of the implementation the two operands are compared to identify denormalization and infinity. Then the two exponents are subtracted to obtain the exponent difference and identify whether the operands need to be swapped using the exponent difference sign. In the second stage the right sifter is used to pre normalize the smaller mantissa. In the third stage addition is done along with the leading one detection. In the fourth stage left shifter is used to post normalize the result. In the last stage the exponent out is calculated and rounding is done. The results are then compared to set overflow or underflow flags.

C. Timing and Area Comparison with Xilinx Intellectual Property

The five stage standard single precision floating point adder is synthesized, placed, and routed for spartan FPGA device using Xilinx ISE 12. The minimum clock period reported by the synthesis tool after placing and routing was 8ns. That means the maximum clock speed that can be achieved for this implementation is 126 MHz. The number of slices reported by the synthesis tool was 394. The maximum delay was shown by third stage where the addition and leading one detection occurs. Inducing registers to implement stages in the design reduces the routing delays significantly compared to one stage pipeline in the previous section. The five stage pipelined standard floating-point adder implementation clock speed is 6.4% better than that reported by Xilinx IP. The area reported for our implementation is 23% better than the Xilinx IP. Due to better slice packing the area occupied by five stage pipelined version of standard adder implementation takes around 27% (147 slices) less than its non pipelined version. The IP doesn't give the algorithm or the internal module implementation or stage placement thus it is hard to compare in detail the reasons behind these numbers. This study is done to mainly to give designers a comparison between our implementation and the IP available for sale. The result shown in figure 18.

6. Conclusions

Floating-point unit is an integral part of any modern microprocessor. With advancement in FPGA architecture, new devices are big and fast enough to fit and run modern microprocessors interfaced on design boards for different applications. Floating-point units are available in forms of intellectual property for FPGAs to be bought and used by customers. However, the HDL for the design is not available to be modified by the customers according to their design needs. Floating-point adder is the most complex component of the floating-point unit. It consists of many complex sub components and their implementations have a major effect on latency and area of the overall design. Over the past two decades, a lot of research has been done by the VLSI community to improve overall latency for the floating-point adder while keeping the area reasonable. Many algorithms have been developed over time. In order to reduce confusions among programmers and vendors, IEEE introduced the IEEE 754 standard in 1985 which standardizes floating-point binary arithmetic for both software and hardware. There are many floating-point adder implementations available for FPGAs but to the best of our knowledge, no work has been done to design and compare different implementations for each sub component used in the floating-point addition for a FPGA device. The main objective of our work was to implement these components and obtain best overall latency for the three different algorithms and provide HDL and discuss solutions to improve custom designs. Standard algorithm consists of the basic operation which consists of right shifter, 2's complement adder, leading one detector, and left shifter. Different implementations for all these various components were done using VHDL and then synthesized for Xilinx spartan FPGA device to be compared for combinational delay and area. The objective was to reduce the overall latency; therefore each sub component is selected accordingly. Standard algorithm is also considered as naive algorithm for floating-point adder and is considered to be area efficient but has larger delays in levels of logic and overall latency. For a Xilinx spartan FPGA device our implementation of the standard algorithm occupied 370 slices and had an overall delay of 31 ns. The standard algorithm was also pipelined into five stages to run at 100 MHz which took an area of 324 slices. The pipelined version of LOD adder was a better choice when compared to Xilinx IP Core [12] running at 22% better clock speed and thus giving a better throughput. The main objective of this research was to develop a design resource for designers to implement floating-point adder onto FPGA device according to their design needs such as clock speed, throughput and area. This kind of work has not been done before, to the best of our knowledge, and we believe it would be a great help in custom implementation and design of floating-point adders on FPGAs.

.

ACKNOWLEDGMENT

Special thanks to Professor h.b.balazadeh, Professor in the Department of Computer Science Engineering, university of tabriz, my advisor for this work, for allowing me to choose such an interesting area of VLSI Design. I am also very thankful to all seniors of VLSI lab in the Electronics and Communications Engineering Department, and thanks to Professor kozeh kanani Dean of department electrical and computer for guide me in this section.

REFERENCES

[1] M.Farmland, "On the Design of High Performance Digital Arithmetic Units, " PhD thesis, Stanford University, Department of Electrical Engineering, August 1981.
[2] P.M.Seidel, G.Even, "Delay-Optimization Implementation of IEEE Floating-Point Addition," IEEE Transactions on computers, pp. 97-113, February 2004, vol. 53, no. 2
[3] J.D.Bruguera and T.Lang, "Leading-One Prediction with Concurrent Position Correction," IEEE Transactions on Computers, pp. 1083–1097, 1999, vol. 48, no.10.
[4] S. F.Oberman, H.Al-Twaijry and M. J.Flynn,"The SNAP Project: Design of Floating-Point Arithmetic Units" Proc. 13th IEEE Symp. on Computer Arithmetic, pp. 156-165, 1997.
[5] Xilinx, http://www.xlinix/com.
[6] L.Louca, T.A.Cook, W.H.Johnson, "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs," FPGAs for Custom Computing, 1996.
[7] W.B. Ligon, S.McMillan, G.Monn, F.Stivers, and K.D.Underwood, "A Re-evaluation of the Practicality of Floating-point Operations on FPGAs," IEEE Symp. On Field-Programmable Custom Computing Machines, pp. 206–215, April 1998.
[8] E.Roesler, B. E. Nelson, "Novel Optimizations for Hardware Floating-Point Units in a Modern FPGA Architecture," Field-Programmable Logic and Applications, pp. 637-646, September 2002.
[9] J.Liang, R.Tessier and O.Mencer, "Floating Point Unit Generation and Evaluation for FPGAs," IEEE Symp. on Field-Programmable Custom Computing Machines, pp. 185-194, April 2003.
[10] G.Govindu, L.Zhuo, S.Choi, and V.Prasanna, "Analysis of High-Performance Floating-Point Arithmetic on FPGAs," International Parallel and Distributed Processing Symp., pp. 149b, April 2004.
[11] Digital Core Design, http://www.dcd.pl/ .
[12] Quixilica, http://www.quixilica.com/
[13] Nallatech, http://www.nallatech.com/
[14] IEEE Standard Board and ANSI, "IEEE Standard for Binary Floating-Point Arithmetic," 1985, IEEE Std 754-1985.
[15] J.Hennessy and D.A.Peterson, Computer Architecture a Quantitative Approach, Morgan Kauffman Publishers, second edition, 1996.
[16] V.G.Oklobdzija, "An Algorithmic and Novel Design of a Leading Zero Detector Circuit: Comparison with Logic Synthesis." IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 124-128, 1994, Vol. 2, No. 1.
[17] Israel Koren, Computer Arithmetic Algorithms, A K Peters, second edition, 2002.
[18] J.D.Bruguera and T. Lang,"Rounding in Floating-Point Addition using a Compound Adder," University of Santiago de Compostela, Spain Internal Report, July 2000.
[19] M.J.Flynn, S.F.Oberman, Advanced Computer Arithmatic Design. John Wiley & Sons, Inc, 2001.
[20] M.J.Flynn, "Leading One Prediction -- Implementation, generalization, and application," Technical Report: CSL-TR-91-463, March 1991.
[21] S.F.Oberman, "Design Issues in High performance floating-point arithmetic units," Technical Report: CSL-TR-96-711, December 1996.
[22] N. Shirazi, A.Walters, P.M. Athanas, "Quantitative analysis of floating point arithmetic on FPGA based custom computing machines". IEEE Symp. on Field-Programmable Custom Computing Machines, pp. 155-163, 1995.

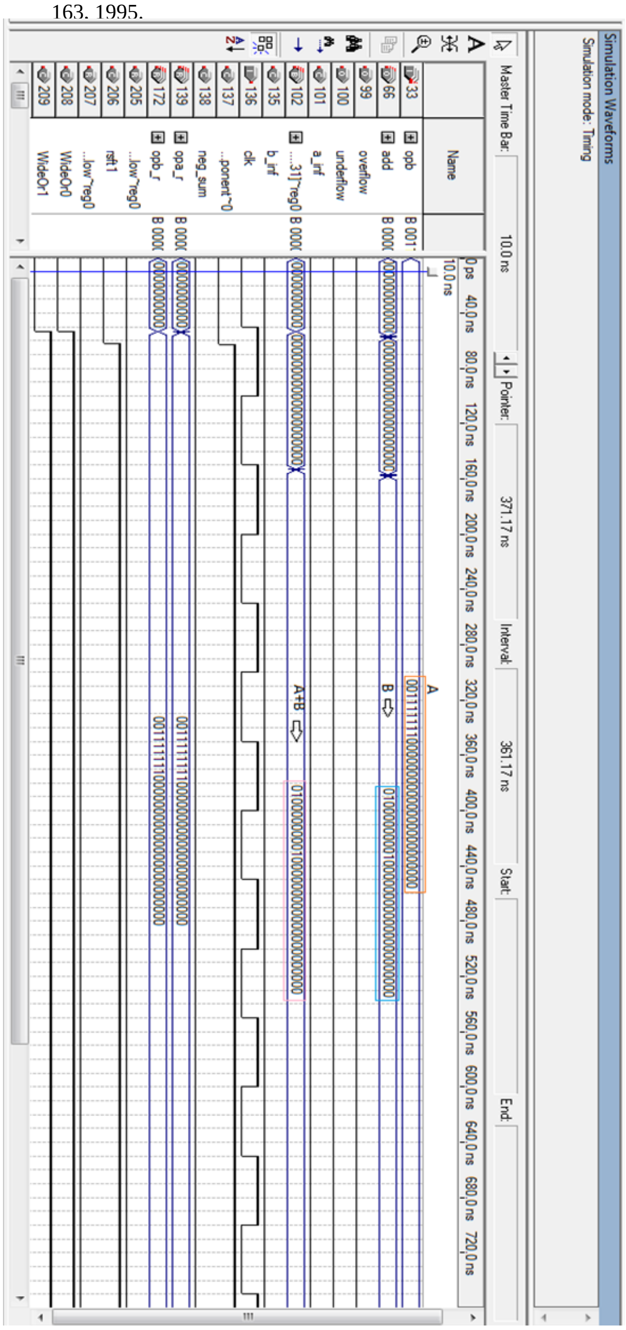

Fig. 18.The Result Of Floating Point Adder.

BIOGRAPHIES

Ali Farmani received his B.Sc. in Electronics Engineering from the Shiraz University of Technology, and M.Sc. degrees in Electronics Engineering from the University of Tabriz, in 2011.respectively. His main research interests are in Nano electronic, VLSI architectures and integrated circuit (IC) design and design digital circuit with VHDL on FPGA and DSP,Cryptography,digital signal processing, image, and video processing,design integrated GPS/INS system.