International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-1 Issue-4, April 2012

Pipelined Floating-Point Arithmetic Unit (FPU) for Advanced Computing Systems using FPGA

Rathindra Nath Giri, M.K.Pandit

Abstract: ***Field Programmable Gate Arrays (FPGA) are increasingly being used to design high-end computationally intense microprocessors capable of handling both fixed and floating-point mathematical operations. Addition is the most complex operation in a floating-point unit and offers major delay while taking significant area. Over the years, the VLSI community has developed many floating-point adder algorithms mainly aimed to reduce the overall latency. An efficient design of floating-point adder onto an FPGA offers major area and performance overheads. With the recent advancement in FPGA architecture and area density, latency has been the main focus of attention in order to improve performance. Our research was oriented towards studying and implementing standard .Our work is an important design resource for development of floating-point adder hardware on FPGAs. This article reports the work done on the design of control path and data path of an optimized 64bit floating-point addition operation using field programmable gate array. At first we selected carry skip adder logic due to its best performance in terms of area, speed and power then we employ common a graph to determine the best mix of cutsets, registors, MUXs for pipelining.***

Keywords: **Data path, control path, VLSI, FPGA, adder logic, floating point addition, optimization, pipelining.**

I. INTRODUCTION

Performance issues in digital systems such as clock skew and its effect on setup and hold time constraints, and the use of pipelining for increasing system clock frequency. This is followed by definitions for latency and throughput with associated resource tradeoffs explored in detail through the use of dataflow graphs and scheduling tables applied to digital signal processing applications.

Improvements in microprocessor performance have been sustained by increases in both instruction per cycle (IPC) and clock frequency. In recent years, increases in clock frequency have provided the bulk of the performance improvement. These increases have come from both technology scaling (faster gates) and deeper pipelining of designs (fewer gates per cycle). In this paper, we examine for how much further reducing the amount of logic per pipeline stage can improve performance. The results of this study have significant implications for performance scaling in the coming decade.[1]

Manuscript published on 30 April 2012.
***** Correspondence Author (s)
Rathindra Nath Giri, Department of electronics and communication engineeringe, Haldia Institute of technology, Haldia, India, +919434452868, (e-mail: rathin.eie@gamil.com).
Prof.(Dr.)Malay Kumar Pandit, Dean of school of engineering and professor of electronics and communication engineering, Haldia Institute of technology, Haldia, India, +919883035230.,
(e-mail: mkp10011@yahoo.com).

II. BACKGROUND

Field Programmable Gate Arrays (FPGAs) have evolved enormously from 10,000 to 10,000,000 logic gates.This density increase, including the overall technology improvements, make FPGAs a good choice to implement DSP applications[1]. FPGAs are an attractive tool for rapid prototyping because of their high programmability and fast reconfiguration. An FPGA is an array of fine-grained configurable logic blocks interconnected in a hierarchical fashion. Commercial FPGAs contain coarse-grained blocks such as memories and multipliers for commonly used primitives to improve efficient for specific functions[2]. A number of applications like space and military where FPGAs are used demand low power features. A major component of FPGA power is the datapath and so there is a scope for reduction in power consumption there. Many applications using FPGAs involve computations that use arithmetic operators like adders, multipliers, etc. These arithmetic operators can be implemented in different architectures. The power consumed by these operators can be decreased by proper selection of a particular operator architecture over the other. The power varies not only because of the different operator architectures but also because of its interconnects, that depend on the FPGA architecture and technology. For this, synthesis tool must have detailed knowledge of parameters like power, delay and area of operators of different implementations for different bit widths. [9]

III. FRAMEWORK

A. *Suitable adder logic*

In electronics, an adder or summer is a digital circuit that performs addition of numbers. In many computers and other kinds of processors, adders are used not only in the arithmetic logic unit(s), but also in other parts of the processor, where they are used to calculate addresses, table indices, and similar. The module generator offer a wide variety of adder architectures that can be selected according to the power and performance needs of the design. The implementations include ripple carry adder, carry skip adder, carry select, Carry Look-Ahead adders and Brent and Kung adder.

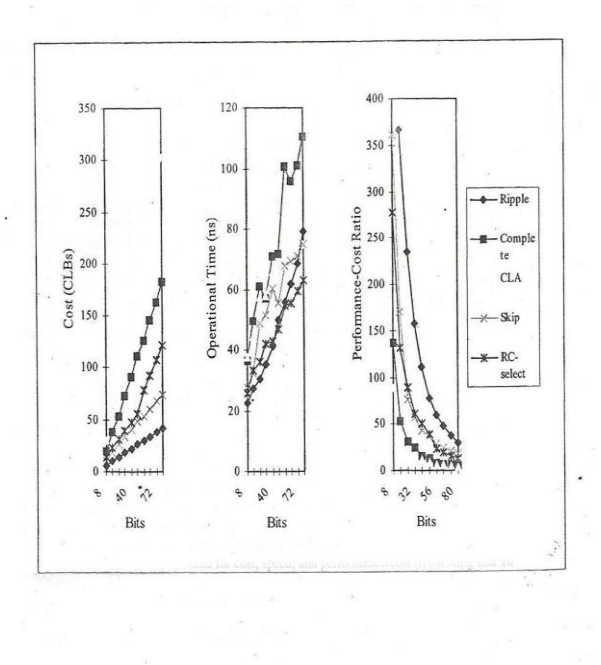

Fig 1. FPGA adder compared for cost speed and performance cost

Figure 1 compares ripple-carry, carry-skip, carry-select adders using the formulas with delay parameters fitted from a Xilinx 4010 FPGA. The result shows that ripple-carry are faster than either carry-skip or carry-select adders for any adder smaller than 48 bits. The carry-skip is the second fastest until additions of 56 bits [3] .

As it is evident from Table 1, the VHDL adder shows the least combinational delay and area. In adders, the delay is mostly offered due to the propagation of carry. While designing custom adders, carry-look ahead adder offers the best delay because the carry is calculated separately looking at the inputs. VHDL adders use the inbuilt carry-chains in CLBs of the Virtex 2p FPGA and provide very small delay and area and thus are chosen for all further adder and subtraction implementations. An 8-bit adder is used to subtract the exponents and the carry out is used to identify if e1 is smaller than e2. If the result is negative, it has to be complemented and a 1 has to be added to it in order to give out the absolute difference.

Table 1. Adder implementation analysis

Adder Type	Combinational Delay (ns)	Slices
Ripple-Carry Adder	15.91	18
Carry-Save Adder	11.951	41
Carry-Look Ahead Adder	9.720	39
VHDL Adder	6.018	8

B.Floating point addition

Floating point addition is the most frequent floating point operation. FP adders are critically important components in modern microprocessors and Digital Signal Processors. The design of Floating Point Adders is considered more difficult than most other arithmetic units due to the relatively large number of sequentially dependent operations required for a single addition. Addition of floating point numbers involves the prealignment, addition, normalization and rounding of significands as well as exponent evaluation. Significand prealignment is a preerequisite for addition. In floating point additions, the exponent of the larger number is chosen as the tentative exponent of the result. Exponent equalization of the smaller floating point number to that of the larger number demands the shifting of the significand of the smaller number through an appropriate number of bit positions. The absolute value of the difference between the exponents of the numbers decides the magnitude of alignment shift. Addition of significands is essentially a signed magnitude addition, the result of which operation is also represented in signed-magnitude form. Signed-magnitude addition of significands can lead to the generation of a carry out from the MSB position of the significand or the generation of leading zeros or even a zero result. Normalization shifts are essential to restore the result of the signed-magnitude significand addition into standard form. Rounding of normalized significands is the last step in the whole addition process. Rounding demands a condition a incrementing of the normalized significand. The operation of rounding, by itself can lead to the generation of a cany out from the MSB position of the normalized significand. That means, the rounded significand need be subjected to a correction shifting in certain situations.

Figure 2. illustrates the block diagram of a floating point adder. The exponent comparator and differencer block evaluates the relative magnitudes of the exponents as well as the difference between them. The significand selector block routes the significand of the larger number to the adder while routing the significand of the smaller number to the Barrel Switch. The Barrel Switch performs the requisite pre-alignment shift (right shift). The sign magnitude adder performs significand addition subtraction, the result of which is again represented in sign-magnitude form, The leading zero counter evaluates the number of leading zeros and encodes it into a binary number. This block, essentially controls normalization shifts. The rounding logic evaluates rounding conditions and performs significand correction whenever rounding is necessary. The correction shift logic performs a right shift of the rounded significand if the process of rounding has resulted in the generation of a carry out from the MSB position of the significand. The exponent corrector block evaluates the value of the exponent of the result. The flag logic asserts various status flags subject to the validity of various exception conditions.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

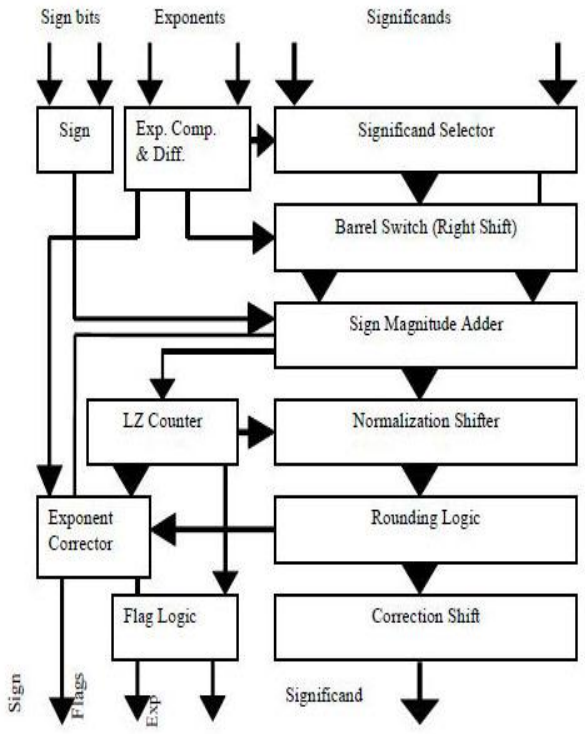

Fig 2. Block diagram of floating point adder

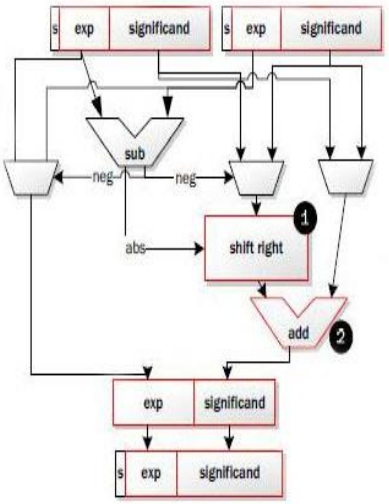

Fig 3. Architecture of floating point adder

C . Problems associated with floating point addition

For the input the exponent of the number may be dissimilar. And dissimilar exponent can't be added directly. So the first problem is equalizing the exponent. To equalize the exponent the smaller number must be increased until it equals to that of the larger number. Then significant are added. Because of fixed size of mantissa and exponent of the floating-point number cause many problems to arise during addition and subtraction. The second problem associated with overflow of mantissa. It can be solved by using the rounding of the result The third problem is associated with overflow and underflow of the exponent. The former occurs when mantissa overflow and an adjustment in the exponent is attempted the underflow can occur while normalizing a small result. Unlike the case in the fixed-point addition, an overflow in the mantissa is not disabling; simply shifting the mantissa and increasing the exponent can compensate for such an overflow. Another problem is associated with normalization of addition and subtraction. The sum or difference of two significant may be a number, which is not in normalized form. So it should be normalized before returning results.[11]

D. Algorithm of floating point addition

Step 1

Compare the exponents of two numbers for (or) and calculate the absolute value of difference between the two exponents. Take the larger exponent as the tentative exponent of the result.

Step 2

Shift the significand of the number with the smaller exponent, right through a number of bit positions that is equal to exponent difference. Two of the shifted out bits of the aligned significand are retained as guard (G) and Round R) bits. So for p bit significands, the effective width of aligned significand must be p + 2 bits. Append a third bit, namely the sticky bit (S), at the right end of the aligned significand. The sticky bit is the logical OR of all shifted out bits.

Step 3

Add/subtract the two signed-magnitude significands using a p + 3 bit adder. Let the result of this is SUM.

Step 4

Check SUM for carry out (Cout) from the MSB position during addition. Shift SUM right by one bit position if a carry out is detected and increment the tentative exponent by 1. During subtraction, check SUM for leading zeros. Shift SUM left until the MSB of the shifted result is a 1. Subtract the leading zero count from tentative exponent.Evaluate exception conditions, if any.

Step 5

Round the result if the logical condition R"(M0+ S'') is true, where M0 and R''represent the pthand (p + 1)st bits from the left of the normalized significand. New sticky bit (S'') is the logical OR of all bits towards the right of the R''bit. If the rounding condition is true, a 1 is added at the pthbit (from the left side) of the normalized significand. If p MSBsof the normalized significand are 1's, rounding can generate a carry-out. in that case normalization (step 4) has to be done again [4].

IV. Data Path and Control Path

A. Pipelining

Pipelining is a well known method for improving the performance of digital systems. Pipelining exploits concurrency in combinational logic in order to improve system throughput. A pipelined machine requires data and control signals at each stage to be saved at the end of every cycle. In the subsequent clock cycle this stored information is used by the following stage. Therefore, a portion of each clock period is required by latches to sample and hold values. In this section, we first vary the pipeline depth of an in-order issue processor to determine its optimal clock frequency. The stop-and-wait problem is not

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

limited It really applies to all links such that the link transmission rate multiplied by the propagation delay is large compared to the frame size. This is sometimes called the bandwidth-delay product or pipe size because it measures the number of bits that can physically b e stored on the physical link. Since the speed of light has been constant for centuries, but transmission rates keep improving, the bandwidth-delay product keeps increasing. For all these reasons, pipelined data link protocols are essential.[8]

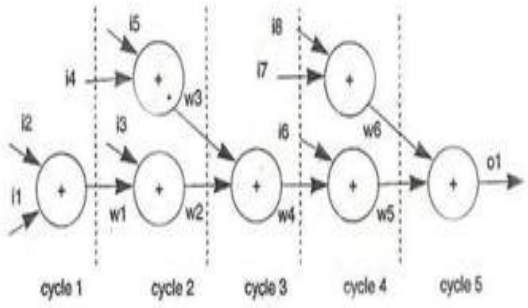

Fig.4 Data flow graph pipelined adder

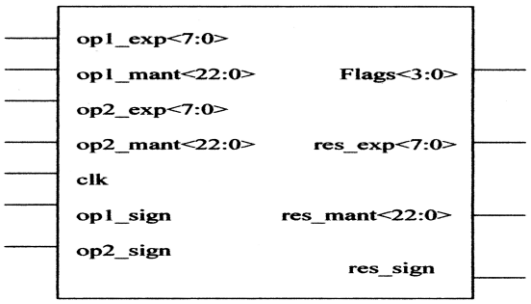

Fig.5 Block diagram of pipelined FP adder

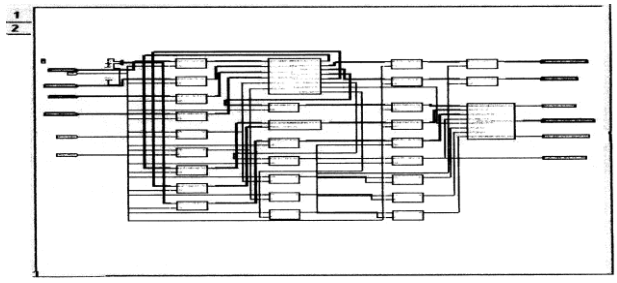

Fig.6 RTL schematic of pipelined FP adder

B. Throughput and Latency

Some programmers write functions that do a small amount of work on a data atom and return a result. They optimize the function to do its task and return as quickly as possible. This is frequently described as the *low-latency* approach to optimization, because it seeks to reduce the function latency -- the time between when a function is called and when it returns. Low-Latency functions are simple and easy to use. Data is traditionally passed by value and functions are typically small. On the other hand, when the overall goal is to complete an operation on your entire data set as quickly as possible, rather than just a small fragment of it, pursue a *high-throughput* design. High throughput designs are those that focus on moving as much data through the calculation as possible in the shortest amount of time. In order to achieve that, any part of the calculation that doesn't need to be repeated for each new piece of data is moved out of the inner loop. Just the essential core operations are repeated for each data. These functions can be complex and are most effective when operating over large data sets [5].

V. IMPROVED FLOATING-POINT ADDER ALGORITHMS

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

The leading one predictor algorithm is pipelined into five stages to be compared with the Xilinx IP Core [12] provided by Digital Core Design. All the key features of the Xilinx IP Core, were implemented except invalid flag. 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 prenormalized the smaller mantissa. In the third stage the addition is done along with leading one prediction. 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.

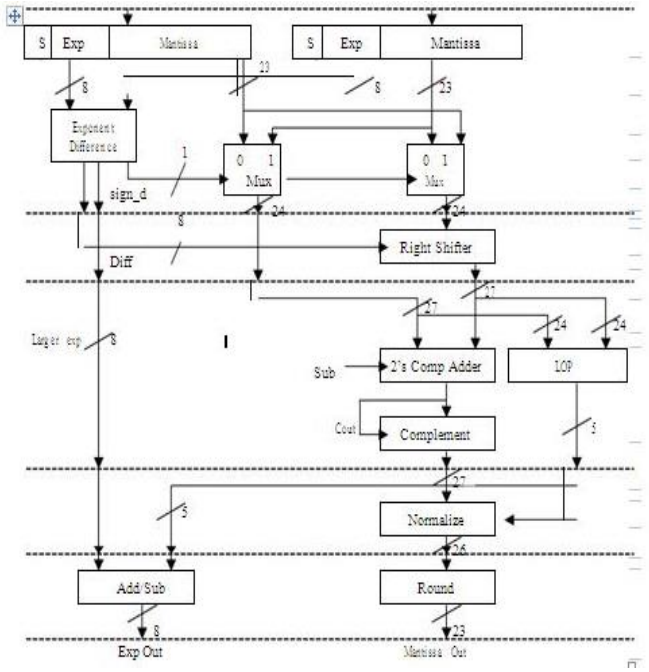

Fig7. Micro-architecture of 5 stage pipeline LOP floating-point adder

B. *Far and Close Data-path Algorithm*

In standard floating-point adder, the critical path in terms of latency is the pre- normalization shifter, the integer addition, leading-one detection, post-normalization, and then the rounding addition. Leading one predictor is used in the LOP algorithm to do the addition and leading one detection in parallel, however, as shown in results it decreases the number of logic levels but doesn't have a very big effect in overall latency if synthesized for a Xilinx Virtex 2p FPGA device. Far and close data-path algorithm adder is designed on the research work based on

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

rounding in floating-point adders using a compound adder.

According to the studies, 43% of floating-point instructions have an exponent difference of either 0 or 1. A leading one detector or predictor is needed to count the leading number of zeros only when the effective operation is subtraction and the exponent difference is 0 or 1, for all the other cases no leading zero count is needed. 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 4-7 shows the flowchart of the far and close data-path algorithm. A description of the algorithm is as follows.

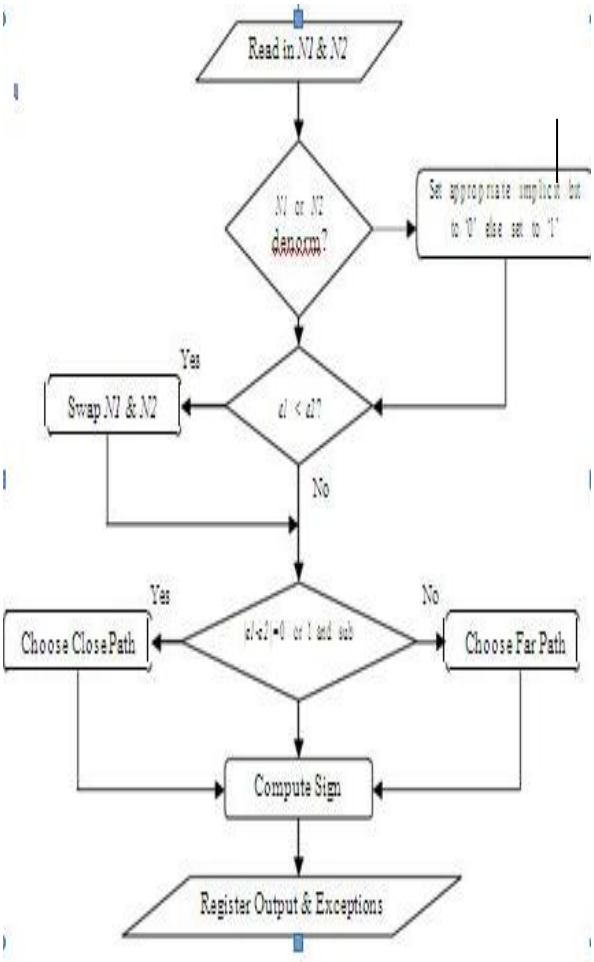

Fig8. Flow chart of far and close floating-point adder

the micro-architecture of the far and close path floating point adder. The exponent difference and swap units for pre-normalization are the same as in the standard or LOP algorithm. The two fractions, exponent difference, and larger exponent are the common inputs to both the paths. Close path is chosen if the exponent difference is 0 or 1 and the effective operation is a subtraction otherwise the far path is chosen. The close and far path will be explained in detail as below.

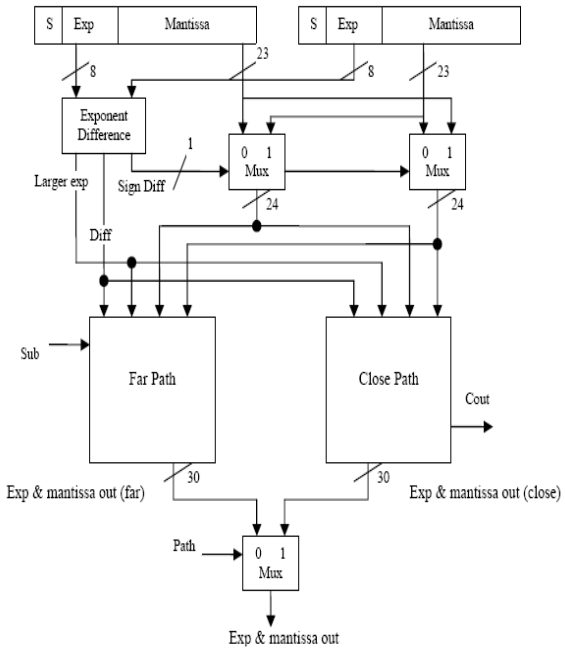

Fig9. Micro-architecture of far and close path floating-point adder

C.Execution problem

A common problem in processor design is that some operations take quite a long time. If you waited for one operation to complete before starting the next one on the same unit, it would be quite slow. A solution to this problem is to break down the calculation into many shorter steps, similar to an automobile assembly line. As the clock advances, intermediate calculations are advanced down the assembly line from one pipeline stage to the next until a finished calculation emerges at the end of the pipe. As a calculation moves from stage 1 to stage 2, another instruction can move in behind it and occupy stage 1. With the next tick of the clock, the first instruction moves to stage 3, the second to stage 2 and a third instruction can begin execution in stage 1 and so forth. Were this process to go on long enough, we would achieve a throughput of one instruction completed per cycle, and the pipelines would be full of data undergoing calculation [6].

EVALUATION

Table 2. Comparison between pipelined and unpipelined FP adder

Results	Unpipelined FP Adder	Pipelined FP Adder
No. of slices	1525 out of 20480(4%)	975out of 20480(4%)
No. of slice flip flops	61 out of 40960(0%)	300 out of 40960(0%)
No. of 4 input LUTs	2783 out of 40960(6%)	1764 out of 40960(4%)

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

No. of bonded IOBs	103 out of 489(21%)	102 out of 489(20%)
Minimum period	92.877ns	11.587ns
Maximum frequency	10.767MHz	80.304MHz
Maximum i/p arrival time before clock	201.521ns	1.504ns
Maximum i/p arrival time after clock	5.690ns	5.727ns

Table 3. Throughput, Latency Estimation

Pipeline Depth (n)	Delay (D) (ns.)	Latency (L) (ns.)	Throughput (T=n/D)	ClockPeriod (P=D/n)
1	3.83	3.83	0.2611	3.83
2	2.10	4.20	0.9524	1.05
3	1.46	4.38	2.056	0.48
4	1.17	4.68	3.418	0.2925
5	0.98	4.90	5.102	0.196

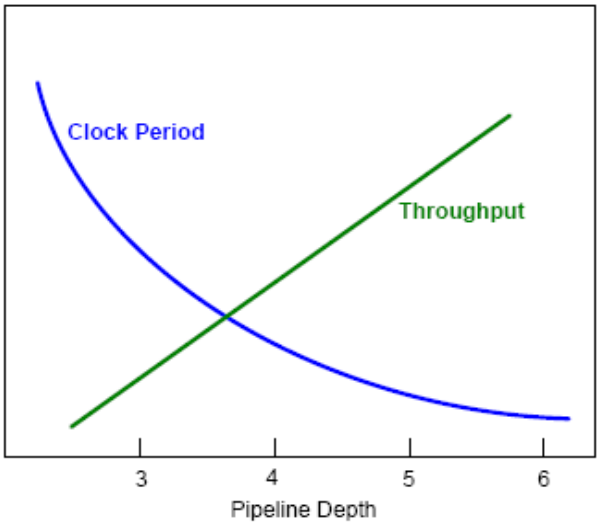

Fig.10 clock period and throughput as a function of pipeline depth.

Table 4. Standard and far and close data-path algorithm analysis

	Clock Period (ns)	Clock speed (MHz)	Area (Slices)	Levels of logic
Standard	27.059	36.95	541	46
F&C path	21.821	45.82	1018	30
% of improvement	+19%	+19%	-88%	+34%

Table 5. 5 stage pipeline LOP implementation and Xilinx IP analysis

	Clock speed(MHz)	Area(Slices)
Xilinx IP[12]	120	510
5 Stage Pipeline LOP	152.555	591
% of improvement	+22%	-15%

Here we have taken five pipelining stages and we determine the corresponding throughput, latency, clock speed. Then we plot a graph to see the changes of these parameters with increase of pipeline depth.

Figure 6.shows how clock period and throughput change with the number of pipeline stages . Throughput increases linearly with pipeline depth.clock period decreases dramatically with the first few pipeline stages.but we add more pipeline stages ,we are subdividing a smaller and smaller clock period and we obtain smaller gains from pipelining . we could ultimately pipeline a machine so that there is a register between every pair of gates,but this would add considerable cost in registers [10].

We have worked with different stages of pipeline and we have seen from the graph that for floating point addition the six stage pipeline is best.

CONCLUSION

When Synthesizing digital circuits different architectures are obtained for different objective functions. A tool that can synthesize automatically a targeted circuit is desire-able. This is because optimum architectures can be selected beforehand and incorporated. This paper proposes a novel methodology to determine optimized FPUs in FPGAs, based on common subgraph extraction. We measured the effects of varying clock frequency on the performance of a superscalar pipeline. The design is. described as graphical schematics and VHDL code. This dual representation is very valuable. as allows for easy navigation over all the components of the units, which allows for a faster understanding of their interrelationships and the different aspects of a Floating Point operation.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

REFERENCES

[1] Irvin Ortiz Flores," Optimizing the Implementation of Floating Point Units for FPGA Synthesis" Electrical and Computer Engineering Department.
[2] ChiWai Yu, Alastair M. Smith, Wayne Luk, *Fellow, IEEE*, Philip H. W. Leong, *Senior Member, IEEE* "Optimizing Floating Point Units in Hybrid FPGAs".
[3] Wayne Wolf, FPGA based system design.
[4] A. J. Al-Khalili," A CAD tool for low power scalable floating-point", Concordia University, Montreal
[5] Ted Williams, "Latency and Throughput Tradeoffs in Self-Timed Speed-Independent Pipelines and Rings", Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University
[6] Justin Davis and Robert Reese, "Finite State Machine Datapath Design, Optimization, and Implementation"
[7] A. J. Al-Khalili, " A CAD tool for low power scalable floating-point adder generator", Concordia University, Montreal
[8] M.S. Hrishikesh, , Norman P. Jouppi_, Keith I. Farkas, Doug Burger_ StephenW. Keckler_ Premkishore Shivakumar. "The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays".
[9] A. Sudnitson, "FINITE STATE MACHINES WITH DATAPATH PARTITIONING FOR LOW POWER SYNTHESIS**",** Tallinn Technical University, ESTONIA
[10] Ted Williams, "Latency and Throughput Tradeoffs in Self-Timed Speed-Independent Pipelines and Rings" Departments of Electrical Engineering and Computer Science Stanford University
[11] Subhash Kumar Shrama, Himanshu Pandey, Shailendra Sahni and Vishal Kumar Srivastava, " Implementation of IEEE-754 Addition and Subtraction for Floating Point Arithmetic Logic Unit ", D.D.U. Gorakhpur University, Gorakhpur, U.P (India)

AUTHOR PROFILE

Rathindra Nath Giril received his bachelor's in electronics and instrumentation engineering in 2009 and pursuing master's degree in electronics and communication engineering in 2012, his specialization is Microelectronics And VLSI Design. He has published 1 international research papers.

Malay K Pandit received his B.E and M. E degrees in Electronics Engineering from Electronics and Telecom Engg Dept, Jadavpur University, India in 1989 and 1991, respectively. He received his PhD from UK's renowned Cambridge University in 1996. He did his post-doc from the Optoelectronics Research Centre, City University of Hong Kong till 2002 where he pioneered the use of polymers for optical waveguide applications. .He then took a corporate career where he worked in a fibre optic company "FONS (I) Ltd" in the domain of optical networking. Now he is a full Professor in the Electronics Engg Dept of the Haldia Institute of Technology where he focuses on embedded systems, including their usage in WDM optical networks. He has more than 50 international publications in this area. Dr. Pandit was awarded the National Scholarship of the Government of India in 1983 and the Nehru Scholarship of the Government of India during his Ph.D. studies.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)