Research Journal of Engineering Sciences ISSN 2278 – 9472
Vol. **1(2),** 1-6, August **(2012)** *Res. J. Engineering Sci.*

VHDL Environment for Floating point Arithmetic Logic Unit - ALU Design and Simulation

Shrivastava Purnima, Tiwari Mukesh, Singh Jaikaran and Rathore Sanjay
[2]Department of Electronics and communication, Shri Satya Sai Institute of technology and Science, Sehore, MP, INDIA

Available online at: www.isca.in
Received 31st May 2012, revised 23rd June 2012, accepted 7th July 2012

Abstract

VHDL environment for floating point arithmetic and logic unit design using pipelining is introduced; the novelty in the ALU design. Pipeling provides a high performance ALU. Pipelining is used to execute multiple instructions simultaneously. In top-down design approach, four arithmetic modules, addition, subtraction, multiplication and division are combined to form a floating point ALU unit. Each module is divided into sub- modules. Two selection bits are combined to select a in the ALU design are realized using VHDL, design functionalities are validated through VHDL simulation. Synthesis and simulation result find out in the Xilinx12.1i platform.

Keywords: ALU-arithmetic logic unit, top-down design, validation, floating point, test-vector.

Introduction

Floating point describes a system for representing numbers that would be too large or too small to be represented as integers. Floating point representation is able to retain its resolution and accuracy compared to fixed point representation. Numbers are in general represented approximately to a fixed number of significant digits and scaled using an exponent. The base for the scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form: Significant digits × $Base^{exponent}$, $S \times B^{e}$

IEEE 754 standard[1] for floating point representation in 1985. Based on this standard, floating point representation for digital system should be platform –independent and data are interchanged freely among different digital systems. Arithmetic logic unit (ALU) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer. Inputs to the ALU are the data to be operated on (called operands) and a code from the control unit indicating which operation to perform. Its output is the result of the computation.

In many designs the ALU also takes or generates as inputs or outputs a set of condition codes from or to a status register. These codes are used to indicate cases such as carry-in or carry-out, overflow, divide-by-zero, etc. Floating point unit also performs arithmetic operations between two values, but they do so for numbers in floating point representation. And the ALU with floating point operations is called a FPU.

Top-down approach (is also known as step-wise design) is essentially the breaking down of a system to gain insight into its compositional sub-systems. In a top-down approach an overview of the system is formulated, specifying but not detailing any first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements. A top-down model is often specified with the assistance of "black boxes", these make it easier to manipulate. However, black boxes may fail to elucidate elementary mechanisms or be detailed enough to realistically validate the model[2].

In order to stimulate a device off board, a series of logical vectors must be applied to the device inputs. These vectors are called test vectors and are mostly used to stimulate the design inputs and check the outputs against the expected values.

A pipeline is a technique used in the design of computers and other digital electronic devices to increase their instruction throughput (the number of instructions that can be executed in a unit of time). The fundamental idea is to split the processing of a computer instruction into a series of independent steps, with storage at the end of each step. This allows the computer's control circuitry to issue instructions at the processing rate of the slowest step, which is much faster than the time needed to perform all steps at once. The term pipeline refers to the fact that each step is carrying data at once (like water), and each step is connected to the next (like the links of a pipe). The origin of pipelining is thought to be the IBM stretch project. Implementing pipeline requires various phases of floating point operations be separated and be pipelined into sequential stages.

We propose VHDL environment for floating point ALU design and simulation. To ease the description, verification, simulation and hardware realization. VHDL is widely adopted standard and has numerous capabilities that are suited for designs of this sort .the use of VHDL for modeling is especially appealing since it provides formal description of the system and allows the use of specific description styles to cover the different abstraction

levels (architectural, register, transfer and logic level) employed in design[2].

Material and Methods

The main objective of this paper is to describes the implementation of pipelining in design the floating -Point ALU using VHDL. The sub objectives are to design a 16-bit floating point ALU operating on the IEEE 754 standard. Floating point representations, supporting the four basic arithmetic operations; addition, subtraction, multiplication and division. Second sub objective is to model the behavior of the ALU design using VHDL.

Specifications for a 16-bit floating-point ALU design- i. Input A and B and output result are 16-bit binary floating point. ii. Operands A and B operate as follows: A (operation) B=results, operation can be addition (+), subtraction (-), multiplication (*), division (/). iii. 'Selection' a 2-bit input signal that selects ALU operation and operate as shown in table-1. iv. Status a 4-bit output signal work as a flag an microprocessor. v. Clock pulse is only provided to the module which is selected using demux. vi. Concurrent processes are used to allow processes to run in parallel.

Table-1
Select ALU operation

Selection	Operation
00	Addition
01	Summation
10	Multiplication
11	Division

Table-2
Select Status

Output	Status
0000	Normal operation
0001	Overflow
0010	Underflow
0100	Result zero
1000	Divide by zero

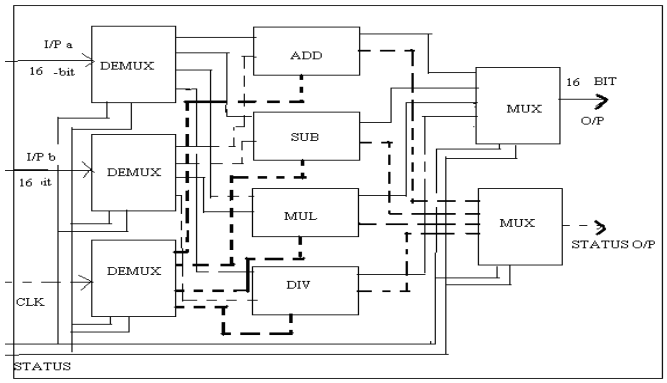

Figure-1
Top level view of the ALU design

ALU is separated into smaller modules: addition, subtraction, multiplication and division, demux and mux. Each arithmetic module is further divided into smaller modules the top level view of figure 1 shows the top level view of the ALU. It consist of four functional arithmetic modules, three demultiplexes and two multiplexers. The demuxs and muxes are used to route input operands and the clock signal to the correct functional modules. They also route outputs and status signals based on the selector pins[3].

After a module completes its task, outputs and status signals are sent to the muxes where they multiplexes with other outputs from corresponding modules to produce output result selector pins are routed to these muxes such that only the output from currently operating functional module is sent to the output port. Clock is specifically routed rather then tied permanently to each module since only the selected functional modules need clock signals. This provides power savings since the clock is supplied to the required modules only and avoid invalid results at the output since the clock is used as a trigger in every process[4].

Pipelining floating point addition module: Addition module has two 16 bit inputs and one16 bit output selection input is used to enable or disable the module this module is further divided into 4 sub modules zero check, align, add_ sub and normalize module.

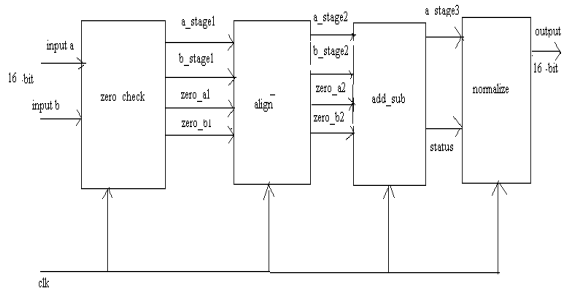

Figure-3
Pipeline floating point addition

Zero check module: This module detect zero operands early in the operation and based on the detection result it has two status signals. This eliminates the need of sub sequent processes to check for the presence of zero operands table 1 summarize the algorithm.

Table-3
Setting Zero Check Bit

I/P b	Zero_a1	Zero_b1
0	1	1
NZ	1	0
0	0	1
NZ	0	

Align module: In this module operations are perform based on status signal from previous stage zero operands are checked in the align module as well this module introduces implied into the operands shown in table 4.

Table-4
Setting of Implied Bit

Zero_a1 xor zero_b1	a_sign	Implied bit for a	Implied bit for b
0	X(do't care)	0	0
1	1	0	1
1	0	1	0

Add_ sub module: This module performs actual addition and subtraction of operands. Firstly operands are checked via the status signals are carried out results are automatically obtained if either of the operand are zero shown in table 3 normalization is needed if no calculation are done here the operation is done based on the science and the relative magnitude of mantissa i.e. summaries in table 5 status signal is set to one is indicate the need of normalization by the next stage[5].

Table-5
check for add_sub module

Zero_a2 &zero_b2	Zero_a1 xor zero_b1	Zero_a2	Result
0	0	X	Perform add_sub
0	1	1	b stage2
0	1	0	a stage2
1	X	X	0

Table-6
Add_Sub Operation

Operation	a_sign xor b_sign	a>b	Result	Sign
a+b	0	X	a+b	+ve
(-a)+(-b)	0	X	a+b	-ve
a+(-b)	1	Yes	a-b	+ve
a+(-b)	1	No	b-a	-ve
(-a)+b	1	Yes	a-b	-ve
(-a)+b	1	No	b-a	+ve

Normalize module: Input is normalize and packed into the IEEE 754 floating point representation if the normalize status signal is set normalization is perform otherwise MSB is dropped.

Pipeline floating point subtraction module: Subtraction module has two 16-bits inputs and one 16-bit output. Selection input is used to enable/ disable the entity depend on the operation. This module is divided further into four sub-modules: zero-check, align, add sub and normalize module. The subtraction algorithm differs only in the add_sub module where the subtraction operator change the sign of the result. The reaming three modules are similar to those in the addition module table 7 and table 8 summarizes the operation.

Table-7
Checks for Add_Sub Module

Zero_a2 &zero_b2	Zero_a2 xor zero_b2	Zero_a2	b_ sign	Result	sign
0	0	X	X	Perform add_sub	NA
0	1	1	0	b_stage2	b_sign=1
0	1	1	1	b_stage2	b_sign=0
0	1	0	X	a_stage2	a_sign
1	X	X	X	0	NA

Table-8
Add_Sub Operation and Sign Fixing

Operation	a_sign xor b_sign	a>b	Result	sign
(-a)-b	1	X	a+b	-ve
a-(-b)	1	X	a+b	+ve
(-a)-(-b)	0	Yes	a-b	-ve
(-a)-(-b)	0	No	b-a	+ve
a-b	1	Yes	a-b	+ve
a-b	1	No	b-a	-ve

Pipelined floating point multiplication module: Multiplication entity has three 16-bit inputs and two 16-bit outputs. Selection input is used to enable/disable the entity. Multiplication module is divided into check-zero, check-sign, add-exponent and normalize–and-concatenate all modules, which are executed concurrently. Status signal indicates special result cases such as overflow, underflow and result zero, in this project pipelined floating point multiplication is divided in to three stages (figure-4). Stage 1 checks whether the operand is zero and report the result accordingly[6]. Stage 2 determines the product sign, add exponents and multiply.

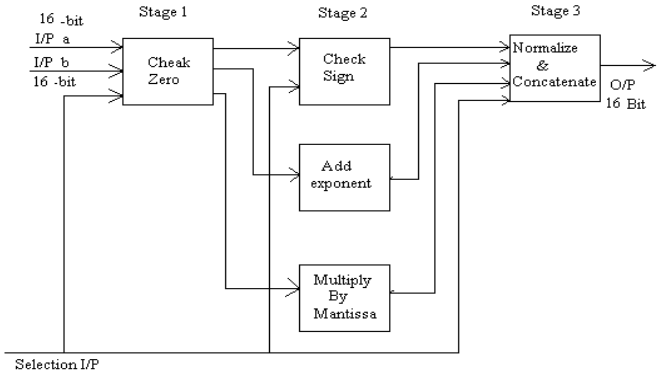

Figure-4
Pipeline structure of multiplication module

Check-zero module: Initially two operands are checked to determine whether they contain a zero. If one of the operand is zero, the zero_flag is set to 1. The output results zero. If neither of them is zero then the inputs with IEEE 754 format is unpacked and assigned to the check sign, add exponent and multiply mantissa modules, the mantissa is packed with hidden bit 1.

Add exponent module: The module is activated if the zero flag is set. Else zero is passed to the next stage and exp_flag is set to 0, two extra bit are added the exponent indicating overflow and underflow.

Multiply mantissa module: In this stage zero_flag is checked first. If the zero_flag is set to 0, then no calculation and normalization is performed. The mant_flag is set to 0 if both the operands are nonzero after the multiplication is done mant_flag is set to 1 to indicate that this operation is executed.

Check sign module: This module determines the product sign of two operands .the product is positive, when the two operands have the same sign; otherwise it is negative. The sign bit are compared using XOR circuit. The sign_flag is set to 1

Normalize and concatenate module: This module checks the overflow and underflow occurs if the 9^{th} bit is 12. Overeflow occurs if the 8^{th} bit is 1. If exp_flag, sign_flag and mant_flag are set, the normalization is carried out. Otherwise, 16-zero bits are assigned to the result. During the normalization operation, the mantissa MSB is 1, hence no, normalization is needed. The hidden bit is dropped and the reaming bit is packed and assigned to the output port. Normalization module set the mantissa MSB to 1. The current mantissa is shifted left until 1 is encountered foe each shift the exponent is decreased by 1, if the mantissa MSB is 1, normalization is completed and first bit is the implied bit dropped. The remaining bits are packed and assigned to the output port. The final normalization product with the correct biased exponent is concatenated with product sign[7].

Pipelined floating point division module: Division entity has three 16-bit inputs and two 16-bit outputs. Selection input is used to enable or disable the entity. Division module is divided into six modules: check zero, align, dividend check sign, subtract exponent, divide mantissa and normalize concatenate modules. Each module is executed concurrently. Status indicates the special cases such as overflow, underflow, and result zero and divides by zero. Figure 5 shows the pipeline structure of the division module.

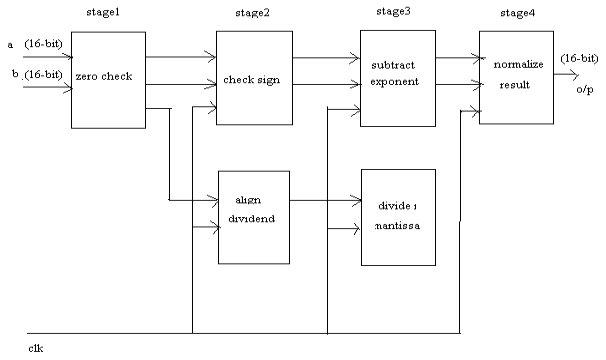

Figure-5
Pipeline structure of the division module

Check-zero module: Initially two operands are checked to determine whether they contain a zero. If one of the operand is zero, the zero_flag is set to 1. The output results zero. If neither of them is zero then the inputs with IEEE 754 format is unpacked and assigned to the check sign, add exponent and multiply mantissa modules, the mantissa is packed with hidden bit 1.

Add exponent module: The module is activated if the zero flag is set. Else zero is passed to the next stage and exp_flag is set to 0, two extra bit are added the exponent indicating overflow and underflow.

Multiply mantissa module: In this stage zero_flag is checked first. If the zero_flag is set to 0, then no calculation and normalization is performed. The mant_flag is set to 0 if both the operands are nonzero after the multiplication is done mant_flag is set to 1 to indicate that this operation is executed.

Check sign module: This module determines the product sign of two operands. The product is positive, when the two operands have the same sign; otherwise it is negative. The sign bit are compared using XOR circuit. The sign_flag is set to 1.

Align dividend module: This module compares both mantissas. If mant_a is greater than or equal to the msant_b then the mant_a must be aligned for every bit right shift of the mant_a mantissa, the mant_a exponent is then increased by 1. This increase may result in an exponent overflow, in this case an overflow flag is set. Otherwise, the process continues with the parallel operation of exponent subtraction and mantissa division. Align_flag is set to 1[7].

Subtract exponent module: This module is activated if the zero flag is set. If not, zero value is passed to the next stage and exp_flag is set to 0. Two extra bits are added to the exponent to indicate overflow. Here two exponents are subtracted. The bias is added back. After this the exp_flag is set to 1.

Divide mantissa module: In this stage, align flag is checked first. If align flag is 0 then no mantissa division is performed mant_flag is set to 0.if both operand are not zero, mant_a is divided by mant_b. In division algorithm, comparison between two mantissa is done by subtracting the two values and checking the output sign.

Results and Discussion

Design is verified through simulation, which is done in a bottom –up fashion. Small modules are simulated in separate test benches before they are integrated and tested as a whole[8]

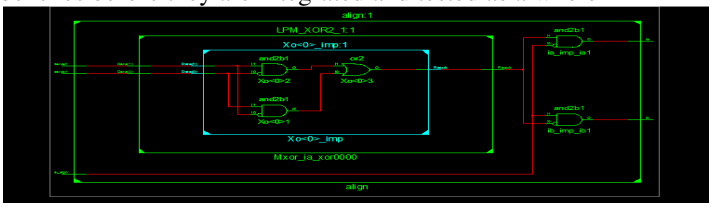

Figure-6
RTL view of align operation

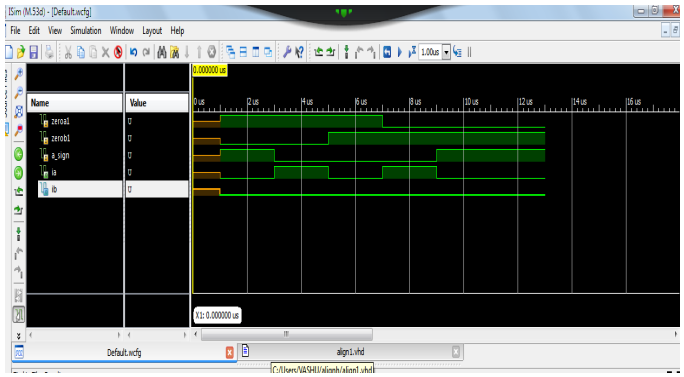

Figure-7
Simulation Result of Align

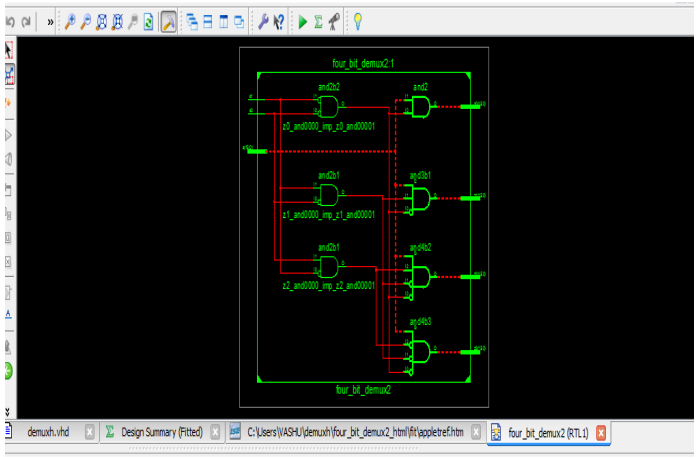

Figure-8
RTL of Demux

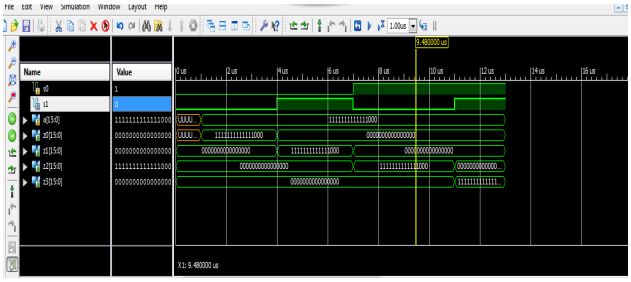

Figure-9
Waveform of demux

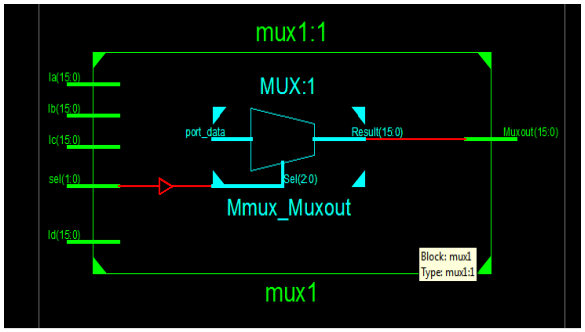

Figure-10
RTL view of Multiplexer

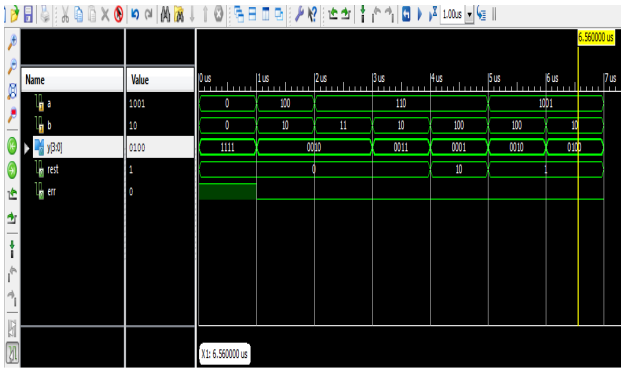

Figure-11
Simulation result of Mux

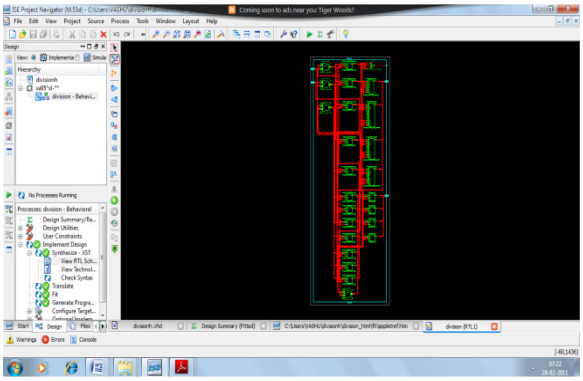

Figure-12
RTL of Division

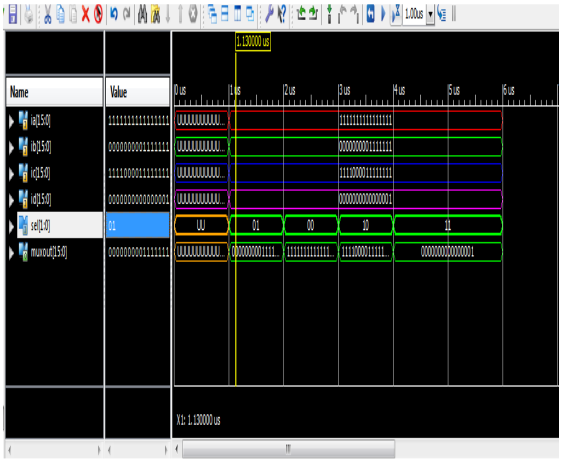

Figure-13
Simulation result of division

Conclusion

By simulation with various test vectors the proposed approach of pipeline floating point ALU design using VHDL is successfully designed, tested and implemented currently, we are conducting further research that consider the further reduction in the hardware complexity in terms of synthesis and fully download the code into Altera FLEXIOK:EPFIOKIOLC,FPGA chip on LC -84 package for hardware[9,10].

References

1. ANSIWEE std 754-1985, IEEE standard for binary Floating-point arithmetic, IEEE New York **(1985)**

2. Daumas M. and Finot C., Division of Floating point Expansions with an application to the computation a Determinant, journal Universal computer Science, **5, (2000)**

3. AMD athlon processor technical brief, Advance Micro DevicesInc, Publication no.22054, Rev.D,Dec **(1999)**

4. Chen S., Mulgeew B. and Grant P.M., A Clustering technique for digital communications Channel equalization using radial basis function Networks, *IEEETran. Neural Networks*, **4**, 570-578 **(1993)**

5. Pipeline Floating Point **ALU** Design using VHDL Mamu Bin Ibne Reaz, MEEE, Md. Shabiul Islam, MEEE, Mohd. S. Sulaiman, MEEE, Multimedia University, ICSE2002 Proc. **(2002)**

6. Floating Point-www.wikipedia.com **(2012)**

7. Proc Design Trade-Offs in Floating-Point Unit, Implementation for Embedded and Processing-In-Memory System, Taek-Jun Kwon, Jeff Sondeen, Jeff Draper SC Information Sciences Institute,4676 Admiralty Way Marina del Rey, CA 90292 U.S.A. **(2005)**

8. Floating Point ALU with parallel paths,Kennth Y.Ng, Saratoga kallif,518452,may **(1990)**

9. VHDL Tutorial,Peter J. Ashenden EDA Consultant, Ashenden Designs Pty. Ltd.,www.ashenden.com. © 2004 by Elsevier Science (USA)

10. Arithmetic and logic design, Wikipedia **(2012)**