Volume 6, Issue 1, January 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Low Power High Speed Arithmetic Unit

[1]Shubhangi M. Joshi, [2]Aditi Prabhune
[1] Ph.D Student, Sathyabhama University,
[1, 2] Assistant Professor, PICT, Pune, Maharashtra, India

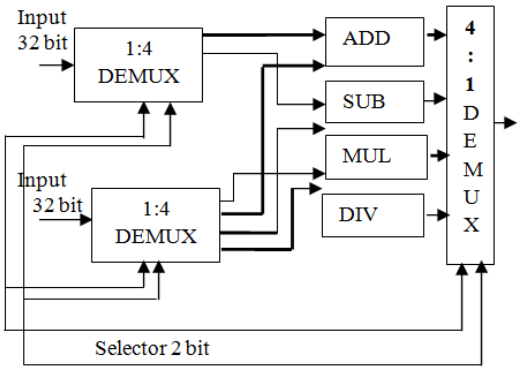
Abstract—Digital signal processors (DSPs) are essential for real-time processing of real-world digitized data, performing the high-speed numeric calculations necessary to enable a broad range of applications – from basic consumer electronics to sophisticated industrial instrumentation. Floating-point processors are useful for computations involving large dynamic range, but they require significantly more resources than integer operations. The rapid advance in Field-Programmable Gate Array (FPGA) technology makes such devices increasingly attractive for implementing floating-point arithmetic. FPGAs offer reduced development time and costs. Moreover, their flexibility enables field upgrade and adaptation of hardware to run-time conditions. In this paper floating point ALU architecture is designed which requires less area and less power.To meet the requirement of real time world the results are optimized using Leading Zero Anticipator. Leading Zero Anticipator is added which will reduce the number of leading zeros of the final output of the addition unit and thus reduce overall power consumption. A error detection logic is proposed in this paper. It works in parallel with adder taking inputs from LZA unlike other existing designs. This resulted in low area and low power design This paper describes Floating Point Arithmetic unit with Reference to IEEE 754 Standard in 32 bit Single Precision Format using VERILOG HDL and simulated usingXilinx simulator covering all functional combinations the designs were mapped on 90nm technology with operating conditions 2.5 V at 25 degree Celsius.

Keywords: FPGA, IEEE, VERILOG, LZA

I. INTRODUCTION

The floating point operations have found intensive applications in the various fields like mathematics and engineering due to its great dynamic range, high precision and easy operation rules. With the increasing requirements for the floating point operations for the high-speed data signal processing and the scientific operation, the requirements for the high-speed hardware floating point arithmetic units have become more and more exigent. The implementation of the floating point arithmetic has been very easy and convenient in the floating point high level languages, but the implementation of the arithmetic by hardware has been very difficult. In this paper a high-speed IEEE754-compliant 32-bit floating point arithmetic unit designed using VERILOG code has been presented.

II. BLOCK DIAGRAM OF FLOATING POINT ARITHMETIC UNIT



FPAU is separated into smaller modules: Addition, Subtraction, Multiplication, Division, Demultiplexers and Multiplexer (selector). Each arithmetic module is further divided into smaller modules that are coded individually. Fig. 1 shows the top level view of the ALU. It consists of four functional arithmetic modules, three Demultiplexers and one Multiplexer. The Demultiplexers and Multiplexer are used to route input operands and the clock signal to the correct functional modules. They also route outputs based on the selector pins.

After a module completes its task, outputs signals are sent to the MUX where they multiplexed with other outputs from corresponding modules to produce output result. Selector pins are routed to these MUX such that only the output from the currently operating functional module is sent to the output port.

'Selection' a 2-bit input signal that selects FPAU operation and operate as shown below: Selections

00 - Addition
01 - Subtraction
10 - Multiplication
11 – Division

III. ALGORITHM FOR ADDITION

- Step 1-Enter two numbers N_1 and N_2. E_1, S_1 and E_2, S_2 represent exponent and significand of N_1 and N_2 respectively.
- Step 2-Is E_1 or E_2 = "0". If yes; set hidden bit of N_1 or N_2 is zero. If not; then check if $E_2 > E_1$, if yes swap N_1 and N_2 and if $E_1 > E_2$; contents of N_1 and N_2 need not to be swapped.
- Step 3-Calculate difference in exponents $d=E_1-E_2$. If d = 0 then there is no need of shifting the significand. If d is more than 0 say y ,then shift S_2 to the right by an amount y and fill the left most bitby zero.

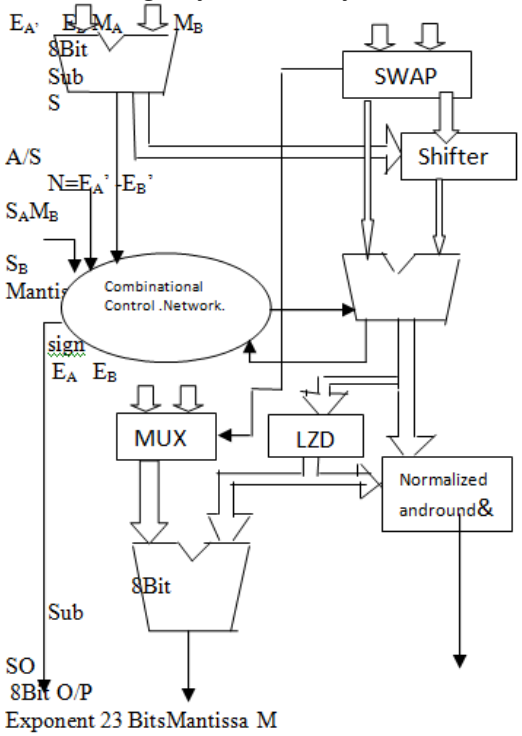


Fig.2: Floating Point Addition/Subtraction Unit

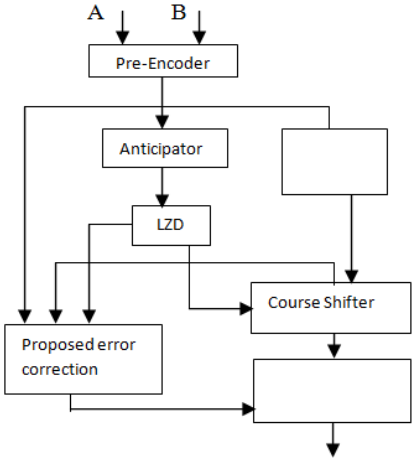
Shifting is done with hidden bit.

- Step 4-Amount of shifting i.e. y is added to exponent of N_2 value. Newexponent value of E_2= (previous E_2) + y. Now result is in normalize form because $E_1 = E_2$.
- Step 5- Check if N_1 and N_2 have different sign.
- Step 6- Add the significand of 24 bits each including hidden bit $S=S_1+S_2$.
- Step 7- Check if there is carry out in significand addition. If yes; then add 1 to the exponent value of either E_1 or new E_2. After addition, shift the overall result of significand addition to the right by one by making MSB of S as 1 and dropping LSB of significand.
- Step 8- If there is no carry out in step 6, then previous exponent is the real exponent.
- Step 9-Sign of the result i.e. MSB = MSB of either N_1 or N_2.
- Step 10-Assemble result into 32 bit format excluding 24th bit of significand i.e. hidden bit.

IV. LEADING ZERO ANTICIPATOR

- In floating point addition unit, adder and normalization decides the critical path delay. By predicting the shift amount prior to the adder's output the delay introduced by the normalization can be reduced. This prediction is done using a technique called Leading Zero Anticipator (LZA). LZA algorithms are divided into exact and inexact categories. Most of the existing algorithms are inexact in nature which predicts the shift amount with a possible error of 1 bit. So, these inexact LZA algorithms need an error detection circuit.
- Leading-Zero Anticipator is a technique that predicts the position of leading digit. It is implemented in parallel with the adder. LZA is followed by Leading-Zero Detector (LZD) which counts the number of leading zeros in predicted string. This is the "shift amount" that is given to normalizing unit. Generally, LZAs are divided into two categories: Exact LZA and Inexact LZA. Exact LZA predicts the "shift amount" precisely with high latency and may come into critical path. Inexact LZA has low latency but predicts the position of leading digit with

error possibility of one bit. So, there is need of using an error detection circuit for an inexact LZA. This error detection circuit corrects the one-bit error occurred in inexact LZA using a one-bit shifter(Fine Shifter) in later stages of normalization.



Optimum IEEE754 format (32Bit)

Block level representation of proposed method in floating point

Anticipator Logic:

Consider two inputs A and B in their magnitude representation

$$A = a_{n-1}.a_{n-2}.\ldots\ldots\ a_1.a_0$$
$$B = b_{n-1}.b_{n-2}.\ldots\ldots\ b_1.b_0$$

The variable C = $c_{n-1}.c_{n-2}.\ldots.c_1.c_0$ is defined as

$$c_i = b_i \quad \text{if addition}$$
$$= b_i' \quad \text{if subtraction}$$

The Pre-encoding block calculates the following propagate, generate and kill strings using the following equations respectively.

$$p_i = a_i \oplus c_i$$
$$g_i = a_i \cdot c_i$$
$$z_i = (a_i \cdot c_i)'$$

Wherep_i,g_i,andz_i represents i[th] bit of P,G,Z strings respectively. Leading zeros occur when sequence p^*gz^* and z^*z' is detected. Where, * indicates zero or more occurrences. Leading ones occur when sequences p^*zg^* and g^*g' are detected.

The indicator function *f* for proposed method is given as

$$f_i = (p_{i+2})' \cdot (z_{i+1} \cdot z_i' + g_{i+1} \cdot g_i')$$
$$+ (p_{i+2}) \cdot (g_{i+1} \cdot z_i' + z_{i+1} \cdot {}_i')$$

Where $f_i$representsi^{th}bit of 'F' string.

For the calculation of f_{n-2} and f_{n-1}

$$P_{n+1} = P_n = P_{n-1}$$
$$Z_n = Z_{n-1}$$
$$g_n = g_{n-1}$$

Since the anticipation Logic is looking for leading zeros and leading ones, it works well for both addition and subtraction irrespective of sign of the result. The predictor string always contains leading zeros either the adder output is positive or negative. LZD outputs "shift amount" by calculating the number of leading zero in predicted string.

V. LEADING ZERO DETECTION

LZD circuit calculates the number of leading zeros in prediction string obtained from LZA. It encodes the number in binary format and is used by coarse shifter for normalization. The method is described below First, the predicted string must be converted into one-hot representation, which is nothing but setting all digits to '0' except the bit in the position of leading digit. For example, the string 00001101 in one-hot representation would be 00001000. In order to do that, a string S has to be used, which is defined as

$$S_i = f_{n-1} + f_{n-2} + f_{n-3} \ldots\ldots\ldots\ldots\ldots + f_1$$

The string, S =sn-1.sn-2...s1.s0 consists of zeros between MSB and leading digit bit and ones from leading digit bit to LSB. For the above example string S will be 00001111. S is then converted into one hot representation using the logic

$$l_i = S_i \,.\, (S_i + 1)'$$
$$l_{n-1} = S_{n-1}$$

The string L = $l_{n-1}l_{n-2}...l_1l_0$represents one-hot representation. Here the logic is looking for the pattern 01, a position of leading digit, marked in the string S. The final encoding of one hot representation into a binary value for operands (A, B) of length 8 bit is done as follows

$$Z_2 = l_3 + l_2 + l_1 + l_0$$
$$Z_1 = l_5 + l_4 + l_1 + l_0$$
$$Z_0 = l_6 + l_4 + l_2 + l_0$$

The binary number, $Z=z_2.z_1.z_0$ is then fed to coarse shifter for normalization. Since, *Z* can count zeros only in range 0-7,"all zero" condition is represented by a flag bit *V* given as

$$V = S_0'$$

VI. ERROR DETECTION LOGIC

This paper discusses two types of error detection logics for comparison with proposed method. In this method, the carry selection unit takes binary number from LZD and checks for carry into leading digit's position in carry process has to wait for the output from adder and carry selection unit lies in critical path. But, the proposed method doesn't come into critical path. It builds a separate logic for error detection. But it uses an excess amount of area and power.

The proposed method's error detection begins by taking inputs from LZA (Anticipator + LZD) and adder as shown in Figure This way, a part hardware of LZA is used which results in reduced area and power consumption. Though, the method outputs its result early compared to proposed method, it will not effect as long as the output is ready before coarse shifter's output is available. So, in proposed method the time gap between error detection block's output and coarse shifter's output is reduced to minimize the area and power consumption.

This method checks for the existence of pattern '*z'*gx*' between the position of leading digit and LSB, '*x*' denotes any string. The string *S*from LZA is taken as input and following operations are done:

$$K = S.\,Z$$
$$J = S.\,G$$

K,*J* strings shows the occurrence of *z*, *g* bits between leading digit's position and LSB respectively.

$$m_i = k_{n-1} + k_{n-2} + \ldots\ldots\ldots + k_i$$

The string *M =mn-1.mn-2...m1.m0* is used as shown below to remove the generate bits between first occurrence of kill bit from leading digit position and LSB.

$$R = M \cdot J$$

Now the *R* string records the occurrences of generate bits which are responsible for carry into leading digit's position. We check for at least one occurrence of generate bits in E string using

$$C = r_{n-1} + r_{n-2} + \ldots\ldots\ldots + r_1 + r_0;$$

c =1 indicates that there is a carry into leading digit's position. But, error correction is also decided by sign of adder's output. Error correction is needed if the adder's output is positive and *c=1* or adder's output is negative and *c=0*.

$$C = c \oplus SUM\,[n-1]$$

Where, *SUM [n-1]* is the MSB of adder's output. C is single bit binary digit, with value '1' indicating a correction of 1 bit, which is done by 1 bit right shift using fine shifter and a value of '0' indicating no need of error correction.

VII. CONCLUSION

A design is based on a newly developed mathematical framework describing the bits of the leading-zero count is presented in this report. We have shown that IEEE single precision floating point arithmetic can be successfully implemented on FPGA.A new error detection logic is proposed in this paper. It works in parallel with adder taking inputs from LZA unlike other existing designs. This resulted in low area and low power design as indicated by simulation results. In this paper Bruguera and Lang's scheme for error detection in LZA and proposed method are implemented for 8 bits.

All the architectures are structurally described using Verilog HDL and simulated using Xilinx simulator covering all functional combinations the designs were mapped on 90nm technology with operating conditions 2.5 V at 25 degree Celsius. The power analysis is done on all designs with 50% toggle rate at 50MHZ frequency. The simulated results are shown. As stated, proposed algorithm uses LZA hardware which contributed in further reduction of area and power.

The chip scope Pro tool allows you to embed the following cores within your design, which assist with on-chip debugging: Integrated Logic Analyzer (ILA), and virtual input/output (VIO) low-profile software cores. These cores allow you to view internal signals and nodes in your FPGA SPARTAN 3E which provide following advantagesReduction of debugging and verification time and remove variation introduced by external debugging solutions.

VIII. PERFORMANANCE PARAMETER

For Xilinx ISE 14.2, as per the implemented design of FPAU, the dynamic power for entire unit according to XPower Analyzer Tool in Xilinx is **5mW** and delay obtained is different units are mentioned below in the following table.

Table: Delay for different Operational Units

Different Operational Unit	Number of Units	Delay(ns)
Addition	1	23.479
Subtraction	1	23.479
Multiplication	1	7.874
Division	1	7.475

APPLICATIONS:-

- Floating Point Unit as a math co-processor in computer system.
- FPU in DSP processor, embedded processor.

REFERENCES

[1] A. Beumont-Smith, N. Burgess, S. Lefrere, and C. C. Lim,"Reducedlatency IEEEfloating-pointstandard dderarchitecture,"inProc. 15thIEEE Symp. Comput. Arithmetic, Jul. 2001, pp. 35–42.

[2] P. M. Seidel and G.Even,"Delay-optimized implementation of IEEEfloating-point addition," IEEE Trans. Comput., vol. 53, no. 2, pp.97–113, Feb. 2004.

[3] S. M. Mueller, C. Jacobi, H.-J. Oh, K. D. Tran, S. R. Cottier, B.W. Michael, H. Nishikawa, Y. Totsuka, T. Namatane, N. Yano, T.Machida, and S. H. Dhong, "The vectorfloating-point unit in a synergistic processor element of a CELL processor,"inProc. 17th IEEESymp.Comput.Arithmetic,Jun. 2005, pp. 59–67.

[4] G. Gerwig and M. Kroener,"Floating-point unit in standard cell designwith 116 bit wide dataflow,"in Proc. 14th IEEE Symp.Comput. Arithmetic, Apr. 1999, pp. 266-273.

[5] N. Ide, M. Hirano, Y. Edo, S. Yoshioka, H. Murakami, A. Kunimatsu, T. Sato, T. Kamei, T. Okada, and M. Suzuoki,"2.44-GFLOPS300-MHz floating-point vector-processing unit for high-performance3-D graphics computing,"IEEE J. Solid-State Circuits, vol. 35, no. 7,pp. 1025–1033, Jul. 2000D. Harris,"An exponentiation unit for an OpenGL lighting engine,"IEEE Trans. Comput. , vol. 53, no. 3, pp. 251–258, Mar. 2004.

[6] S. Sudharsanan and M. Sinnathamby,"Support for variable length decode on embedded processors,"inWorkshop Media Signal Process.Embed. Syst. SoCs , Sep. 2004, pp. 33–51.

[7] N. Weste and D. Harris, CMOS VLSI Design: A Circuits and SystemsPerspective. Boston, MA: Addison Wesley, 2005.

