APPROACH TO PERFORM COMBINATIONAL DIVIDER BASED FLOATING POINT CALCULATIONS USING VHDL COMPONENT

Syed Tahir Hussain Rizvi, Muhammad Yaqoob Javed, Amer Saeed and Haleema Asif
Department of Electrical Engineering, University of Central Punjab, Lahore, Pakistan
tahir.rizvi@ucp.edu.pk

ABSTRACT *- In this paper, a synthesizable model to support floating point calculations is described using combinational divider. This model is suitable to implement trigonometric equations in FPGA devices. Combinational Division Algorithm, approach to use floating point numbers and implementation of look-up tables to provide values of trigonometric function (cosine, inverse cosine etc) in Very High Speed Integrated Circuits Hardware Description Language is briefly introduced. Along the model, test bench waveform is generated for functional verification. Spartan 3 devices are used for implementation. Maximum delay and resource utilization are compared from accuracy point of view (value of up to how many decimal points are required).*

Index Terms *-VHDL, Floating Point Numbers, Combinational Divider, FPGA, Timing Analysis*

I. INTRODUCTION

To control any operation in real-time environment, a number of complex algorithms have to be performed repeatedly which involves use of non-algebraic and arithmetic operations (like addition, multiplication, division, cosine inverse, square roots etc) [1]. To find solution of such algorithms, massive computational power is required.

Concurrent Motion control system can be used to accelerate speed of such computations. Field Programmable Gate Arrays Kits can execute multiple processes concurrently to perform high speed computations. Due to this ability, FPGAs are in great demand. Algorithms designed in Hardware Description Languages (HDL) can be directly implemented on FPGA kit. FPGA is a complete package of logic blocks. Connectivity of these logic blocks to achieve desired function is defined according to code written in HDL. All blocks work concurrently which make the FPGA best choice for faster execution of processes. In FPGA, large size values can be processed due to support of arbitrary length of data arrays. FPGA does not support floating point numbers, but floating point Intellectual Property core based FPGA can be used to achieve accurate results, but tradeoff is between accuracy and cost.

In the widely used HDL languages (VHDL and Verilog), division operation is not synthesizable. Division is basic arithmetic operation required in computation of different of parameters. In power systems, if it is required to compute power factor of a system, then it can be find using (1)

$$\text{Power Factor} = \text{True Power}/\text{Apparent Power} \quad (1)$$

In above equation, it is required to divide two quantities which are not supported in VHDL. Further, if it is required to find phase angle, then formula is

$$\theta = \text{Cos}^{-1}(\text{Power Factor}) \quad (2)$$

Main problem is to deal with floating numbers obtained by division operation and to find cosine inverse which is not supported in VHDL [2][3].

Similarly, in robotics, Inverse Kinematics algorithm is a mathematical tool to compute movement of a robotic structure. If point in Cartesian coordinate (targeted location) and lengths of mechanical links are known, angle of joints required to reach at targeted position can be calculated using inverse kinematics algorithm. Equation (3) is for robotic arm having two joints, where x and y are coordinates/position where arm has to move, l_1 and l_2 are lengths of two links and θ_2 is the angle of second joint. Similarly an equation of θ_1 can be used to find rotational angle of joint 1.

$$\theta_2 = \text{Cos}^{-1}((x^2 + y^2 - l_1^2 - l_2^2)/(2\, l_1^2 l_2^2)) \quad (3)$$

This equation requires division, support of floating point numbers and inverse trigonometric function which are not supported by VHDL [4]. Similarly, numerous equations require all these functions to be performed.

So, in this paper, a synthesizable model of divider that can be extended up to n-bits is presented. This model gives remainder and quotient which are used by a second synthesizable model for providing floating point number. This model provides floating number representation which can be used in look-up tables to provide values of trigonometric function (cosine, inverse cosine etc).

II. THE DIVIDER MODEL IN VHDL

Very High Speed Integrated Circuits Hardware Description Language (VHDL) is used to describe hardware in terms of software. Any digital circuit can be implemented using text based description. VHDL is concurrent language, so mathematical calculations and algorithms can be executed in parallel to speed up the processes. VHDL does not have built-in synthesizable functions of division , square root, cosine and tangent inverse.

Synthesizable model of divider comes from algorithm which is explained in [2][5]. Simple comparison technique is used to perform division operation as shown in TABLE I.

First thing to consider is that number of bits of dividend, divisor and outputs (Quotient and Remainder) should be the same. In this case, dividend, divisor and outputs (Quotient and Remainder) are of 4 bits. If decimal value of dividend is 11 and decimal value of divisor is 3, so for this particular case values of quotient and remainder would be 3 and 2 respectively.

Comparison is performed on shifted version of divisor. Divisor is shifted left to make length of 2n-1 bits (7 bits = (2(4) - 1)).

If

Dividend = 11 = 1011
Divisor = 3 = 0011

Because

Dividend = n bits = 4 bits
Divisor = 2n-1 bits = 7 bits (shift left 2n-1 times)
N steps = 4 steps

Both inputs (dividend and divisor) are compared with each other. If dividend is less than divisor, value of quotient is 0. If dividend is greater than divisor, value of quotient would be 1 and divisor would be subtracted from dividend. This new result would replace the value for next comparison. This complete operation would be completed in n steps, if inputs are of 4 bits, then answer would be achieved after 4 iterations. After last iteration, updated value of dividend would have remainder and all bits received from comparison would make value of quotient.

TABLE I
COMBINATIONAL DIVIDER OF 4 BITS

Dividend	Comparison	Divisor	Quotient	Operation on Dividend
1011	<	0011000	0	None
1011	<	0001100	0	None
1011	>	0000110	1	Dividend - Divisor
0101	>	0000011	1	Dividend - Divisor
0010 (Remainder)			0011 (Quotient)	

This technique is used to implement 10 bits of division in VHDL. As shown in test bench waveform of VHDL model in Fig. 1, if 147 is divided by 22, so remainder is 15 and quotient is 6. Similarly, if 70 is divided by 80, so 70 is less than 80, it can't be divided, so quotient is 0 and remainder is 70.

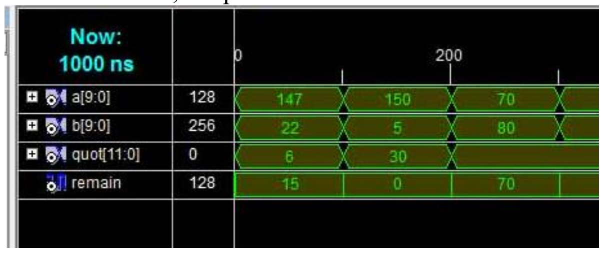
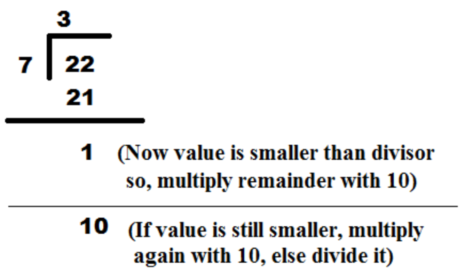


Fig. 1 Verification of Division Operation in Test bench Waveform

III. FLOATING POINT REPRESENTATION MODEL

Synthesizable model of the floating point representation comes from hand-written division. Above defined divider model is used to get values of quotient and remainder, these value can be used to extract value after decimal point. If 22 is divided by 7, remainder is 1 and quotient is 3. Now, this remainder can be used further to take digits after decimal point as shown in Fig 2.

Now value of remainder is 1 that is smaller than divisor 7, it can't be directly divided, so we have to multiply remainder with 10, if value is still smaller multiply again with 10, now division of this value provides the first digit after decimal point.


3
7
22
21
1 (Now value is smaller than divisor so, multiply remainder with 10)
10 (If value is still smaller, multiply again with 10, else divide it)

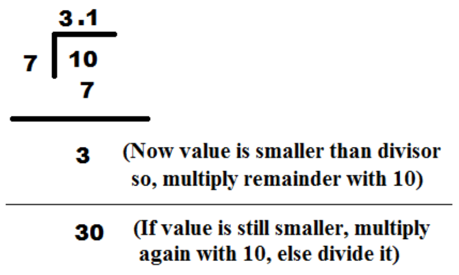

3.1
7
10
7
3 (Now value is smaller than divisor so, multiply remainder with 10)
30 (If value is still smaller, multiply again with 10, else divide it)

Fig. 2 Algorithm to find Digits after Decimal Points

Description of VHDL component for floating point numbers is:

```
entity floating is
Port (a: in STD_LOGIC_VECTOR (9 DOWNTO 0);
b: in  STD_LOGIC_VECTOR (9 DOWNTO 0);
int: out STD_LOGIC_VECTOR (11 DOWNTO 0);
d1:out STD_LOGIC_VECTOR (11 DOWNTO 0));
end floating;
architecture Behavioral of floating is
component divider is
PORT (dividend, divisor: IN INTEGER RANGE 0 TO 4095;
quotient: OUT STD_LOGIC_VECTOR (11 DOWNTO 0);
remainder: OUT INTEGER RANGE 0 TO 4095;
err : OUT STD_LOGIC);
end component;
signal quot: STD_LOGIC_VECTOR (11 DOWNTO 0);
signal remain, quot1,remain1:  INTEGER RANGE 0 TO
4095;
begin
h11: divider port map (
                      dividend =>conv_integer(a),
                divisor =>conv_integer(b),
                      quotient =>int,
                     remainder =>remain
                    );
process(remain)
       variable ten:INTEGER RANGE 0 TO 10:=10;
       variable temp:INTEGER RANGE 0 TO 4095:=0;
       begin
              if(remain<conv_integer(b)) then
                     temp:=remain *ten;
              end if;
       quot1<=temp;
end process;
h22: divider port map(
                      dividend =>quot1,
                      divisor  =>conv_integer(b),
                      quotient =>d1,
                     remainder =>remain
                      );
end Behavioral;
```

It can be verified using Testbench waveforms shown in Fig. 3, that after division, integer part is saved in a variable named "int" and first decimal value is saved in variable d1. If 22 is divided by 7, so answer is "int.d1" means 3.1

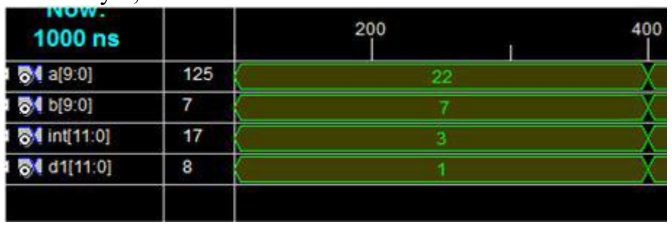


Fig. 3 Floating Point Representation upto First Digit

These steps can be repeated to get floating point value of desired length. In Fig. 4, it can be viewed that if 22 is divided by 7, so answer is “int.d1d2d3d4d5” means 3.14285

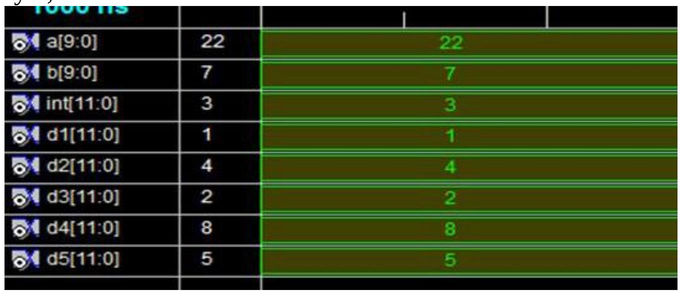


Fig. 4 Floating Point Representation up to Five Digits

All these value can be concatenated in VHDL and look-up table of these values can be implemented in VHDL.

If we concatenate first two digits, for example, d1 is 2 and d2 is 5, they combine to form 0.25, if it is required to find their cosine inverse [COS-1(0.25) = 75], so its look-up table can be implement in VHDL as described through this code.

```
float<=(d1(3 downto 0)) & (d2(3 downto 0)) ;
with float select
 cos_inv<=
"01001011"  when   "0010 0101",
```

"with/select" structure is used in VHDL to apply condition and assign output to final variable depending on matched condition.

In command `with float select,` float is the variable in which our floating digits are combined in bits and of this we want to find cosine inverse.

In command `cos_inv <= "01001011" when "0010 0101",` on right side are the bits on which we want to apply condition (it is basically value of float) or of whom we want to find cosine inverse. In this case, that is `"0010 0101"` = 25 in hexa decimal representation means 0.25 which is achieved through proposed combinational division based float point component.

In command `cos_inv <= "01001011" when "0010 0101",` on left side are the bits of results which is basically answer of cosine inverse [COS-1(0.25) = 75]. That is `"01001011"` = 75 in decimal.

Similarly a complete look up table of cosine inverse or required operation can be defined.

Similarly, all decimal points can be concatenated to provide accurate results as shown in Fig. 5.

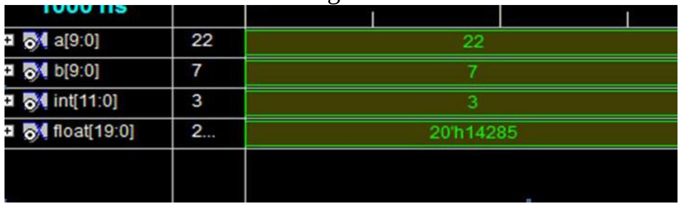


Fig. 5 Concatenated Digits after Decimal Point

IV. RESULTS OF IMPLEMENTATION

Above described model of floating point representation is implemented in Spartan 3 FPGA kit (XC3S200-FT256). Results of implementation are given below in Table II where used slices, critical path delay between input and output and number of Look-up Tables are compared for increasing number of digits after decimal point. These Results are obtained from Synthesis report generated from ISE 9.1i.

TABLE II
COMPARISON OF RESULTS

Number of Floating Digits	Maximum Delay	Look-Up tables Used	Slices Used
1	65.060 ns	474/3840 = 12 %	264/1920 = 13%
2	67.104 ns	731/3840 = 19 %	406/1920 = 21%
3	67.304 ns	988/3840 = 25 %	548/1920 = 28%
4	67.505 ns	1245/3840 = 32 %	690/1920 = 35%
5	67.705 ns	1502/3840 = 39 %	812/1920 = 42 %
8	68.307 ns	2273/3840 = 59 %	1260/1920 = 65 %
11	68.908 ns	3044/3840 = 79 %	1686/1920 = 87 %

Accuracy of 11 decimal points can be achieved using 1686 slices out of 1920 slices, so nearly 87% of resources are utilizing. By using larger FPGA like XC3S1500, this model can be suitable for implementation of floating representation. If same code is executed in XC3S1500, it will utilize only 12% of resources because its total slices are 13312. So tradeoff is between accuracy and resources used but this solution can be implemented on cheaper hardware kits.

V. CONCLUSION

This paper introduces a synthesizable model to support floating point calculations. This is a VHDL component which can be extended to increase accuracy of design. Implementation results upto 11 decimal places are performed. Instead of using intellectual property core based FPGA, algorithm is designed and implemented on cheaper FPGA kit. Accuracy of result can be increased easily by just minor modification but tradeoff is between accuracy and resources of hardware.

REFERENCES

[1] R.R. Singh ; A. Tiwari; V.K. Singh and G.S.Tomar, “VHDL Environment for Floating Point Arithmetic Logic Unit -ALU design and Simulation,” *International Conference on Communication Systems and Network Technologies* : 469- 472 (2011)

[2] V. A. Pedroni, *Circuit Design with VHDL*, 2nd ed., MIT Press, Cambridge, Massachussetts (2004)

[3] K. Ch. Chang, *Digital System Design with VHDL Synthesis: An Integrated Approach*, 1st ed., J. Wiley Inc. (1999)

[4] N. Sorokin, “Implementation of High-Speed Fixed-Point Dividers OnFPGA,” *Journal of Computer Science & Technology*, **6**(1): 8 – 11 (2006)

[5] Z. Fedra and J. Kolouch, “VHDL Procedure for Combinational Divider,” in *34th International Conference on Telecommunications and Signal Processing (TSP)*: 469 – 471 (2011)