Design and Implementation of Binary to IEEE754 64-bit Floating Point Converter using Verilog

Srinivas v[1] Harshitha B[2]
[1]M Tech. Student [2]Assistant Professor
[1]Electronics [2]Dept. of E&C
[1,2]B M S College of Engg, Bangalore, India.

Abstract— Floating point numbers are used in many applications due to their dynamic representation capabilities in digital signal processing. The main objective of the work is to design and implement a binary to IEEE 754 floating point converter for representing 64 bit double precision floating point values. The converter at the input side of the floating point unit module helps to improve the overall design and reduces the complexity of manually converting the binary numbers into IEEE 754 format numbers. Hence the method solves the problems in occurrence of human errors to an extent in direct feeding of IEEE 754 format numbers to the FPU module.

The modules are written using Verilog and are then synthesized for Xilinx Virtex 5 FPGA using Xilinx Integrated Software Environment(ISE) design suite 14.2
Key words: Floating point, IEEE754, Double Precision.

I. INTRODUCTION

The demand for floating point arithmetic operations in most of the commercial, financial and internet based applications is increasing day by day. Hence it becomes essential to find out an option to feed binary numbers directly as input for these applications. This helps in saving time and is much easier. In the current scenario, this is not possible, because, in the floating point unit, inputs should be given in IEEE 754 format i.e[3]. the binary inputs cannot be given as such, but it needs to be converted to the sign, exponent and mantissa form, about which, will be described in detail later. Hence the implemented binary to floating point converter for double precision bits will solve this issue to an extent.

The fundamental difference between fixed and floating point digital signal processors (DSPs) is their respective numeric representation of data. While fixed point hardware performs strictly integer arithmetic, floating point DSPs support integer or real arithmetic, the latter normalized in the form of scientific notation. A 64-bit, binary floating point DSP, supporting industry-standard, double precision operations, provides greater accuracy and greater precision than fixed point devices due to its wider word width, exponentiation and exact internal representations of data. Fixed point devices had to implement real arithmetic indirectly through software routines which add algorithmic instructions and development time, while with floating point format, real arithmetic could be coded directly into hardware operations. So, this thesis emphasizes on utilizing the capabilities of floating point format. The binary input given will range from 0-2048 bits, which is the maximum input range that can be provided which will satisfy the exponent range in the 64 bit IEEE 754 double precision format.

The document is organized as follows: Section II summarizes the important aspects of IEEE 754 single precision format. Section III describes binary to floating point conversion. Section IV presents the results. Section V concludes the thesis.

II. IEEE FLOATING POINT REPRESENTATION

The Institute of Electrical and Electronics Engineering (IEEE) issued 754 standards for binary floating point arithmetic in 1985[1,2,4,6].This standardization was needed to eliminate computing industry's arithmetic vagaries. Later revisions were made to the existing standard in 2008.There are five basic formats-three binary floating point formats and two decimal formats.

The first two binary formats are called 'Single precision' and 'Double precision respectively. IEEE double precision format alone is considered in this work. Hence it is described below.

In IEEE 754-2008, the 64-bit, base 2 format is officially referred to as binary 64. Double precision format uses 1-bit for sign bit, 11-bits for exponent and 52-bits to represent the fraction as shown in Fig 1.

Sign	Exponent	Mantissa
63	62.....52	51.....0

Fig 1: IEEE 754 double precision format

The double precision floating point number is calculated as (-1)s x 1.F x 2(E-1023). Sign bit determines the sign of a number, which is either 0 for a non-negative number or 1 for a negative number. For IEEE double precision format, a bias of 1023 is added to the actual exponent.

The IEEE 754 standard specifies some special values like positive infinity, negative infinity, positive zero, negative zero and Not a Number (NaN).The standard also specifies the following rounding modes also like: Round to nearest, ties to even; Round to nearest, ties away from zero; Round toward positive infinity; round toward negative infinity and round toward zero. These special cases are considered in this work.

III. BINARY TO FLOATING POINT CONVERSION

Our proposal is to convert the floating point number into a simple yet quite precise integral representation and perform the calculations on the same, followed by the final conversion of the output into its expected floating point result format.

The floating point data is inputted in two parts. The first part is a 64 bit binary value of the integer part of the floating point operand and other is a 64 bit binary value of fractional part of the floating point operand. This is done because Verilog cannot deal with floating point numbers. So we need to consolidate the two parts (integral and fractional) of the operand into a single 64 bit effective operand.

3.1 Conversion of floating point number into a novel integral representation shown in the following algorithm.
Step 1: The sign bit (63rd bit) of the input integer part becomes the sign bit of the effective operand.
Step 2: Then the position of 1st significant 1 is searched in the input integer part from RHS. This position is stored.
Step 3: All the bits from this position to the end of the input integer part (i.e. till the 0th bit) is taken and inserted into the effective operand from its 62nd bit onward. (This step stores the actual useful bits of the integer part as not all the 64 bits are used to accommodate the integer part.)
Step 4: If there are still positions in the effective operand that are not filled, then it is filled with the bits from the input fractional part from its MSB down to the number of bits equal to places left to be filled. (This step stores the just requisite number of bits from the fractional part to complete the 64 bit representation).
This can be explained with the help of an example.
Float_op_int=64'b0000000000000000000000000000000000001010100011010100001110000
Float_op_frc=64'b1111111111111000000000011111111100000000000000000000000000000000
Step 1: Assign output[63] = Float_op_int[63]
Step 2: Pos of 1st 1 from LHS of Float_op_int = 25(pos counted from RHS)
Step 3: Assign output = Float_op_int[25:0]
Step 4: Remaining bits left to be assigned in remaning = 64-26-1 = 34
Step 5: output[33:0] = Float_op_frc[63:30]
Output=0101010001101010000111000001111111111111000000000011111111100.
So, basically our technique gives preference to the fractional part for smaller numbers and the integer part for larger ones thus keeping intact the effective precision of the floating point number.

The next step is to convert the output (here the effective operand into the IEEE specified format.

IEEE754 double precision can be encoded into 64 bits using 1 bit for the sign bit (the most significant i.e. 63rd bit), next 11 bits are used for the exponent part and finally rest 52 bits are used for the mantissa part.
This conversion can be done using the below algorithm.
Step1: Sign bit of the binary number becomes the sign bit (63rd bit) of the IEEE equivalent.
Step 2: 62nd bit to11th bit of the binary number becomes the mantissa part of the IEEE equivalent.
Step 3: The exponent part is calculated by subtracting the position of the 1st one obtained in the algorithm described in section 3.1.
Step 4: A bias of 1023 is added to the above exponent value.
Algorithm for convert to IEEE form.
This can be explained with the help of an example.
Output=0101010001101010000111000001111111111111000000000011111111100.
Pos = 25 (from above calculation)
Step 1: op_ieee [63] = output[63]
Step 2: op_ieee [62:52] = 29-pos+1023 = 1027
Step3:op_ieee[51:0]=0101000110101000011100000111111111111100000000001111
(bits that follows the 1st 1 from LHS of output)

S E
(0101000110101000011100000111111111111100000000001111).
M

IV. Simulation Results

The design was simulated using Modelsim SE 6.1 and was targeted towards Vertex 5 FPGA. The device used is xc5vlx30-3ff324. Simulation result is shown in the fig.2 and table.1 shows the device utilization summary.

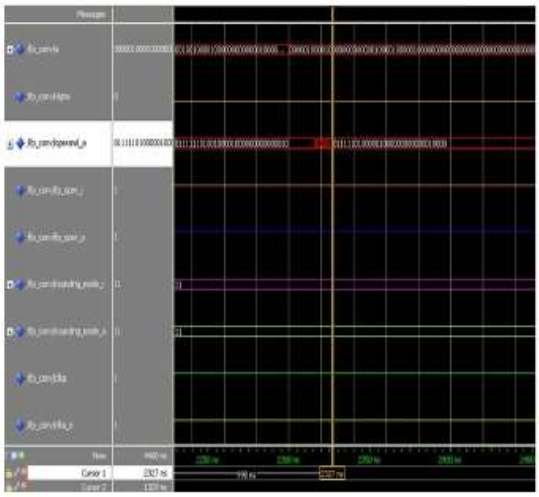

Fig. 2: Simulation result obtained in Modelsim

The simulated results are well matched with manual calculations and hence verified.

Logic Utilization	Used	Available	Utilization
Number of slice latches	62	19,200	1%
Number of 4 input LUTs	2406	19,200	11%
Number of occupied slices	1426	6912	12%
Number of slices containing only related logic	831	831	100%
Total number of 4 input LUTs	2406	19,200	11%
Number of bonded IOBs	328	404	73%
Number of GCLKs	1	4	25%
Number of GCLKIOBs	1	4	25%

Table1. Device Utilization Summary

V. Conclusion

Simulation and Synthesis of double precision binary to floating point converter which is IEEE754 compliant is carried out in this work. The converter block is designed to convert binary inputs into IEEE754 64bit format. Which will be given as inputs to the floating point unit. The design

was simulated using Modelsim SE 6.1 and was targeted towards Vertex 5 FPGA.

REFERENCES

[1] J. D. Bruguera and T. Lang. "*Leading one prediction with concurrent position correction"* IEEE Transactions on Computers,48(10):1083.1097, Oct 1999.
[2] A. Y. Duale, M. H. Decker, H.-G. Zipperer, M. Aharoni, and T. J. Bohizic."*Decimal _floating-point in z9: An implementation and testing perspective"*. IBM Journal of Research and Development, 51(1/2):217.228, 2007.
[3] W. Haller, U. Krauch,T. Ludwig, and H. Wetter. "Combined *binary/decimal adder uni"t*.U.S. Patent 5,928,319, July 1999.
[4] E. Hokenek and R. K. Montoye. *"Leading-zero anticipator (LZA) in the IBM RISC System/6000 _floating-point execution unit"*. IBM Journal of Research and Development, 34(1):71.77, 1990.
[5] Charles Farnum, "*Compiler Support for Floating-Point Computation*" Software Practices and Experience," pp. 701-9 vol. 18, July 1988.
[6] D. Goldberg, "*What every computer scientist should know about floating-point Arithmetic*," pp. 5-48 in ACM Computing Surveys vol.23-1 (1991).

