ISSN 2231-346X (Print)

JUSPS-A Vol. 33(4), 42-50 (2021). Periodicity-Monthly

Section A

ISSN 2319-8044 (Online)

JOURNAL OF ULTRA SCIENTIST OF PHYSICAL SCIENCES

An International Open Free Access Peer Reviewed Research Journal of Mathematics

website:- www.ultrascientist.org

Estd. 1989

Development of Library Components for Floating Point Processor

SUBHASH KUMAR SHARMA[1], SHRI PRAKASH DUBEY[2] and ANIL KUMAR MISHRA[3]

[1]Department of Electronics, M.G.P.G. College, Gorakhpur (India)

[2,3]Department of Physics , M.G.P.G. College, Gorakhpur (India)

[1]Corresponding Author Email: sksharma13@yahoo.co.uk

http://dx.doi.org/10.22147/jusps-A/330402

Acceptance Date 13th June, 2021, Online Publication Date 21st June, 2021

Abstract

This paper deals with development of an n-bit binary to decimal conversion, decimal to n bit binary conversion and decimal to IEEE-754 conversion for floating point arithmetic logic unit (FPALU) using VHDL. Normally most of the industries now a days are using either 4-bit conversion of ALU or 8-bit conversions of ALU, so we have generalized this, thus we need not to worry about the bit size of conversion of ALU. It has solved all the problems of 4-bit, 8-bit, 16-bit conversions of ALU's and so on. Hence, we have utilized VHSIC Hardware Description Language and Xilinx in accomplishing this task of development of conversions processes of ALU.

Key words : Binary, Decimal conversion, Floating point representation, IEEE-754 to Decimal and Decimal to IEEE-754 Conversion, Simulations and Device Utilization.

Introduction

In this paper there are different types of Conversion Process and Different types of Software are used to convert the number .In this paper floating Point Processor is used to represent IEEE-754 to Decimal and IEEE-754 and decimal to IEEE -754 . Here a new design and implementation of a 32-Bit decimal floating point also presents. And here the sequential Architecture of the Proposed 32-Bit Converter is Synthesized with On Xilinx Virtex –II Pro and Virtex-4 FPGA devices. All these are used in Floating Ponit Processor. An Arithmetic and Logic Unit (ALU) is a combinational circuit that performs logic and arithmetic micro-operations on a pair of n-bit Operands. Here we reduced the complexity of problems using Latest software after that using these Software. we can synthesize a processor which troubleshoot our technical problems.

Converting Binary to Decimal :

A simple technique for converting binary to decimal is to rewrite the binary in terms of powers of two then adds the terms. Starting from the right most bit, the weight of each bit is 1, 2, 4, 8, etc doubling the value for each bit. Example: Convert 01011011 to decimal

01011011 = 0 x 128 + 1 x 64 + 0 x 32 + 1 x 16 + 1 x 8 + 0 x 4 + 1 x 2 + 1 x 1 = 64+16 + 8 + 2 + 1 = 91

Converting Decimal to Binary:

The technique for converting binary to decimal can be "reversed" to obtain a technique for converting *decimal to binary*. The idea is to subtract out powers of two from the decimal expansion while keeping track of which powers of two can be subtracted out. Do the following:

1. List the powers of two *right to left* until you reach the first power of two that exceeds the decimal number being converted.

2. Put a 0 over the first power of two that exceeds the decimal number (indicating that the decimal value is too small to contain this power of two). The next lower power of two (to the right) will be smaller than the decimal value. Put a 1 over this power of two and *subtract* the power of two from the decimal value.

3. Move right to the next smaller power of two. If this value is larger than the remainder, put a 0 over the power of two; otherwise put a 1 over the power of two and subtract it from the reminder.

4. Continue until all powers of two have a 0 or 1 over them. This binary integer equals the decimal number.

Example: Convert 91 to binary

1. First we list the powers of two right to left until we exceeded 91 (128 > 91). Put a 0 over 128

```
 0
128   64   32   16   8   4   2   1
```

2. Since 64 is less than 91 we put a 1 over the 64 and subtract it from 91

```
 0     1
128   64   32   16   8   4   2   1  , 91 - 64 = 27
```

3. Continue with the remainder 27. Since 32 is larger than 27, it gets a 0. 16 is less than 27 so we put a 1 over the 16 and subtract it from 27

```
  0    1   0    1
128   64  32   16   8   4   2   1
27 - 16 = 11
```

4. Continue. From 11 we can subtract 8, 2, and 1.

```
  0    1    0    1   1   0   1   1
128   64   32   16   8   4   2   1
 11 - 8 = 3; 3 - 2 = 1; 1 - 1 = 0
```

Therefore 01011011 binary equals 91 decimal. We can check you work by using addition of powers of two to convert 01011011 to 91 decimal.

Literature review :

Many scientific and engineering applications heavily rely on numerical computation. Since such applications demand a large range of representable numbers, almost all numerical computation uses floating-point arithmetic. The IEEE (Institute for Electrical and Electronics Engineers) published in 1985, the IEEE-754 standard for binary floating-point arithmetic[4]. This standard specifies the representation of floating-point numbers as well as the behavior of the floating-point operations performed on these numbers. It was followed two years after by the IEEE-854, radix-independent standard whose major goal was to deal with decimal arithmetic.

The Standard "IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008, New York:" IEEE, Inc.[1], 2008 describes interchange and arithmetic methods and formats for binary and decimal floating-point arithmetic[6,12], in computer programming environments. This standard specifies exception conditions and their default handling. An implementation of a floating-point system conforming to this standard may be recognized entirely in software, entirely in hardware, or in any combination of software and hardware.

The normative part of this standard, numerical results and exceptions specified for different operations in are uniquely determined by the, sequence of operations, values of the input data and destination formats, all under user control. Jain, Jenil, and Rahul Agrawal *et al.*[2] this paper presents design of high speed floating point unit using reversible logic.

In recent nanotechnology[5]. Programmable reversible logic design is trending as a prospective logic design style for implementation and quantum computing with low impact on circuit heat generation. There are various reversible implementations of logical and arithmetic units have been proposed in the existing research, but very few reversible floating-point designs has been designed. Floating-point operations are used very frequently in nearly all computing disciplines. Here in this research paper every type of conversion is done using synthesis and Simulation Using Latest software, Binary to Decimal conversion **Vertex 2Pro,** Decimal to Binary and Decimal to IEEE-754 for **Vertex 4 is used.** and other Software viz, **Model Sim,** Xilinx[9] **Virtex – II Pro**. The code we developed to design this converter, conceptually contains something different and interesting. Floating-point units (FPU) can be implemented in various ways, and the architecture of an FPU has a great effect on its overall performance, area and power dissipation. Arithmetic processor is one of the major fields where we can apply this floating point unit. There we call it **floating point processor** which does computations in accordance with the IEEE-754 floating-point standard. Also hardware implementation of this arithmetic unit is supposed to be done and it is expected that this will significantly accelerate a wide variety of applications. The presented architecture, however, can be optimized to achieve a faster speed or occupy a smaller area.

Floating Point Representation: IEEE 754 Binary Floating Point is a 32-bit representation (for single precision, 64 bits are used for double precision) for floating point numerals. The 32-bit representation consists of three parts. The first bit is used to indicate if the number is positive or negative. The next 8 bits are used to indicate the exponent of the number, and the last 23 bits are used for the fraction. Converting decimal digits to IEEE binary floating point is a little tricky. The purpose of this article is to outline a simple method for completing this conversion.

The first step in the conversion is the simplest. This is determining the first bit. If the decimal digit is positive then this bit is 0, if the decimal digit is negative then this bit is 1.

The next eight digits are used to express the exponent, which we'll figure out last. The final 23 digits are used to express the fraction. This gives you 2^23 precision. The first thing to do is convert our decimal digit to binary, ignoring any positive or negative signs. For instance, if our original number were:

-210.25

We first need to convert 210.25 to binary. It is easiest to focus on the integer part of the number first, then the decimals. 210 is 11010010 or 128+64+16+2, otherwise expressed as:

1*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 0*2^0

Next we need to convert the decimal part. To do this we have to convert the number into a binary sequence that equals $a*2^{-1} + a*2^{-2}$ and so on. This series basically means 1/2 + 1/4 + 1/8 + 1/16 and so on. Luckily .25 is 1/4 and so this is easy to calculate. It is .01 or $0*2^{-1} + 1*2^{-2}$. So our final number is 11010010.01. The next step is to normalize this number so that only one non zero decimal place is in the number. To do this you must shift the decimal place 7 positions to the left. The number 7 becomes important so we note it. This process leaves us with the number 1.101001001 which is the fraction that is represented in the last 23 bit places in the 32-bit binary.

However, because of the rules used for conversion the first digit of any number will always be one. Because we know this number there is no need to represent it. Thus the number we will represent is 101001001. This is then padded with 0's to fill in the full 23 bits - leaving us with 10100100100000000000000.

So we now have the first bit, and the last 23 bits of the 32-bit sequence. We must now derive the middle 8 bits. To do this we take our exponent (7) and add 127 (the maximum number you can express with 8 bits (2^8-1 or the numbers 0 to 127)) which gives us 134. We then express this number as an 8-bit binary. This is 10000110 (or 1*2^7 + 1*2^2 + 1*2^1 or 128+4+2). Now we have the middle bits.
Taken as a whole our bit sequence will be:
1 10000110 10100100100000000000000

We can convert this bit sequence back into a number by reversing the process. First we can note by the leading 1 that the number is negative. Next we can determine the exponent. 10000110 are binary for 2+4+128 or 134. 134-127 is 7, so the exponent is 7.

Finally, we take the last 23 digits, convert them back into the original fraction (adding the preceding 1.) to get: 1.101001001

Moving the decimal place to the right by 7 (corresponding to the exponent) we get:
11010010.01

This binary is equal to 128+64+16+2 + 1/4 or 210.25. Once we apply the negative sign (indicated by the leading bit set to 1) we get our original number:
- 210.25

Formula for converting an IEEE-754 number to decimal :

$$(1-2*s) * (1+f) * 2^{e-\text{bias}}$$

Where s: sign value in decimal ,*f*: mantissa value in decimal, e: exponent value in decimal
bias: bias value 127 (single precision) or 1023 (double precision)

Example IEEE-Decimal conversion :

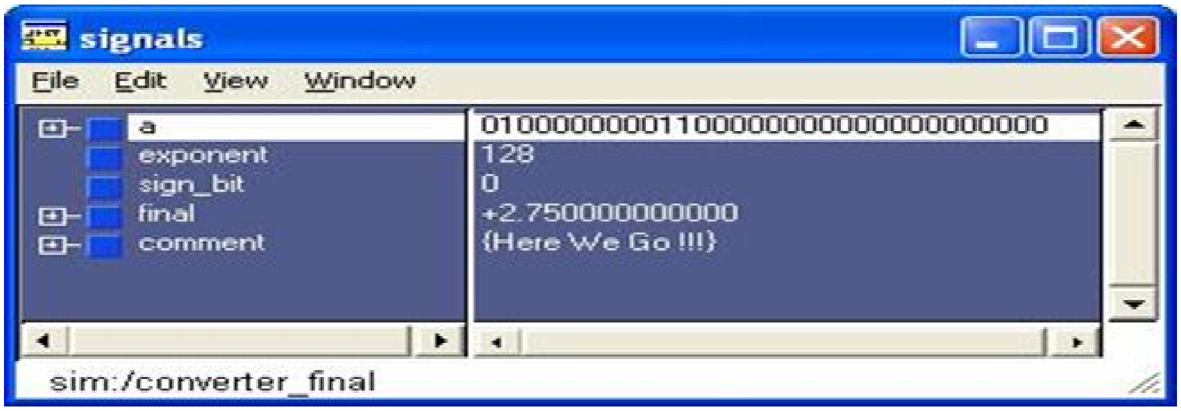
Let's find the decimal value of the following IEEE 0 10000000 01100000000000000000000 First the sign bit s is 0.
The e field contains 10000000 = 128.
The mantissa *f* is 0.011000… = 0.375.
Then just plug these decimal values of s, e
And *f* into our formula. $(1 – 2*s) * (1 + f) * 2^{e\text{-bias}}$
This gives us
$(1 – 2*0) * (1 + 0.375) * 2^{128-127} = (1.375 * 2^{1}) = 2.75$.

Result and Discussion

Simulation and Synthesis Report:

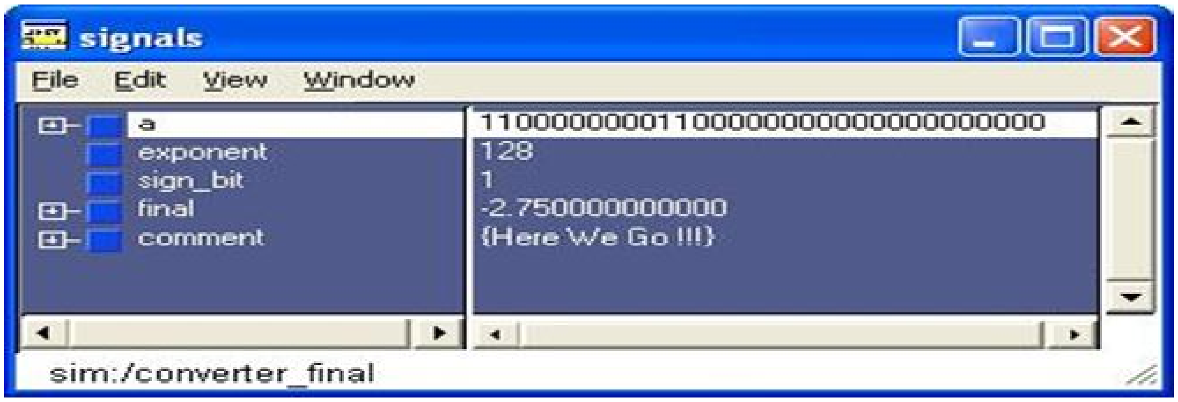
1. a =01000000001100000000000000000000

Exponent = 128, sign bit = 0, Final = +2.750000000000 (observed), Comment = {Here We Go!!!}
Required= (1-2*0)*(1+0.375)*(2^ (128-127))= 2.75, Error = 0%



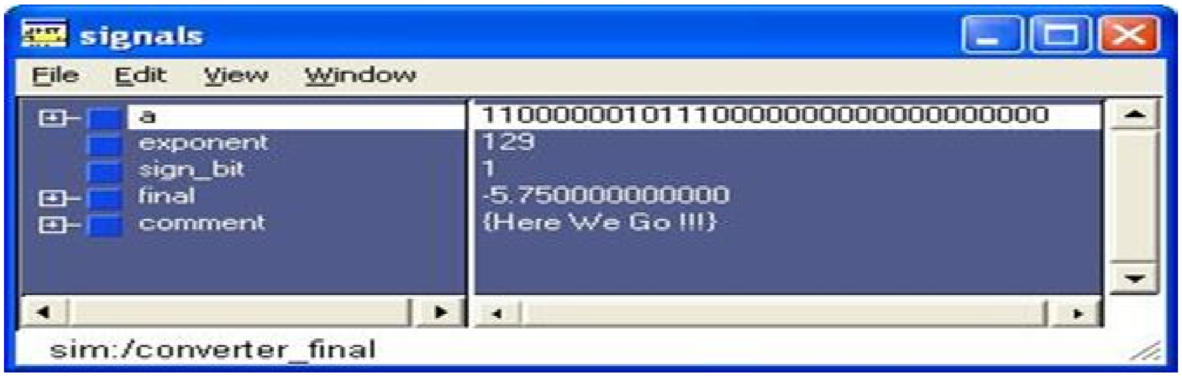
```
a          01000000001100000000000000000000
exponent   128
sign_bit   0
final      +2.750000000000
comment    {Here We Go !!!}
```

2. a = 11000000001100000000000000000000
Exponent = 128, sign bit = 1, Final = -2.750000000000 (observed), Comment = {Here We Go!!!}
Required= (1-2*1)*(1+0.375)*(2^ (128-127))=-2.75, Error = 0 %



```
a          11000000001100000000000000000000
exponent   128
sign_bit   1
final      -2.750000000000
comment    {Here We Go !!!}
```

3. a = 11000000101110000000000000000000
Exponent = 129, sign bit = 1, Final = -5.750000000000 (observed), Comment = {Here We Go!!!}
Required= (1-2*1)*(1+0.4375)*(2^ (128-129)) = -5.75, Error = 0 %



```
a          11000000101110000000000000000000
exponent   129
sign_bit   1
final      -5.750000000000
comment    {Here We Go !!!}
```

Synthesis Report:
For Converting an IEEE-754 number to decimal

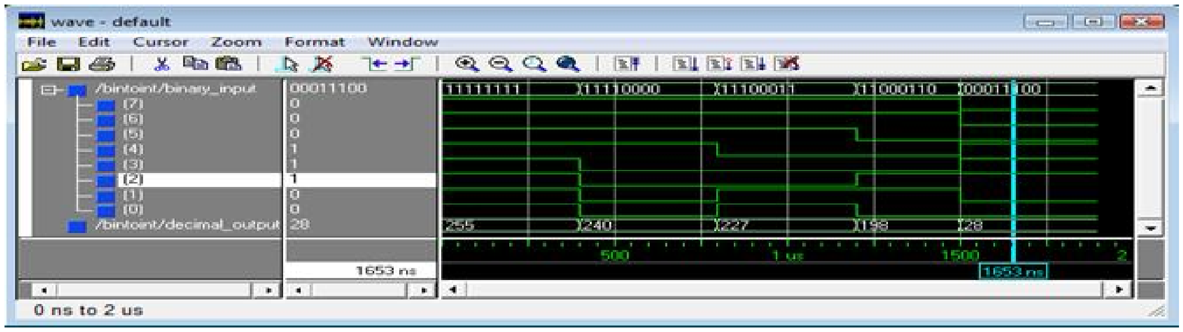
Device Utilization for 2VP4ff672

Resource	Used	Avail	Utilization
IOs	263	348	75.57%
Global Buffers	0	16	0.00%
Function Generator	820	6016	13.63%
CLB Slices	410	3008	13.63%
Dffs or Latches	8	7060	0.11%
Block RAMs	0	28	0.00%
Block Multipliers	2	28	7.14%

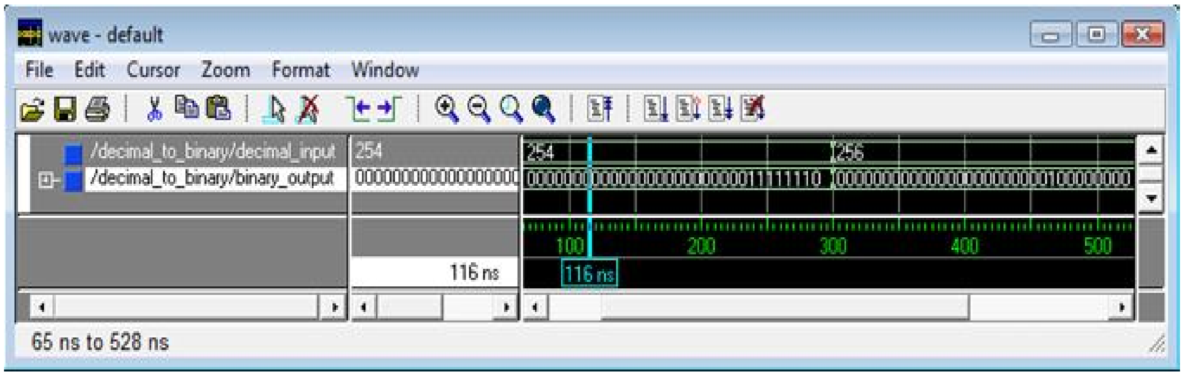
Simulation Results:

1. **n bit Binary to Decimal Conversion:**
2. Input1

Binary input = 11111111, 11110000, 11100011, 11000110, 00011100 all are 8 bit.
Required Decimal out = 255, 240, 227, 198, 28 all are decimal numbers.
Obtained IEEE 754 out =255, 240, 227, 198, 28 all are decimal numbers.

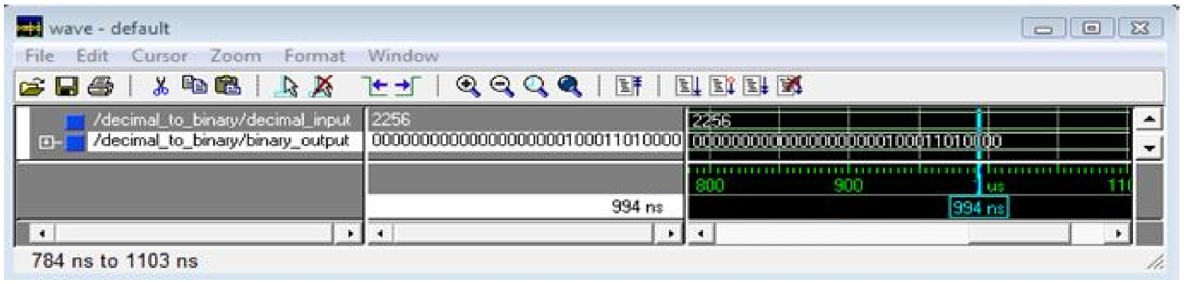


2. Decimal to n bit Binary Conversion: Input1, Decimal input = 254 is decimal number.
Required Binary out = 00000000000000000000000011111110.
Obtained Binary out = 00000000000000000000000011111110.



Input2 : Decimal input = 2256 is decimal number.
Required Binary out = 00000000000000000000100011010000.

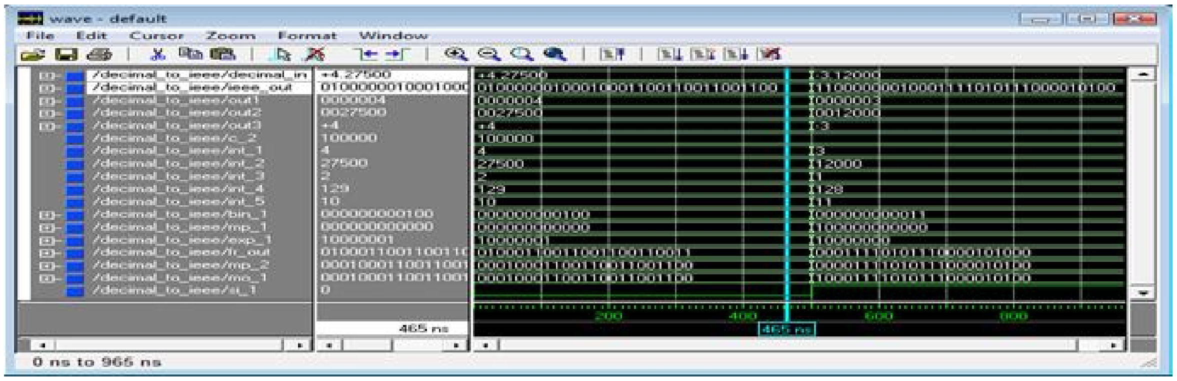
Obtained Binary out = 00000000000000000000100011010000.



```
/decimal_to_binary/decimal_input   2256                               2256
/decimal_to_binary/binary_output   00000000000000000000100011010000   00000000000000000000100011010000
                                   994 ns                             800   900   us   11
                                                                      994 ns
784 ns to 1103 ns
```

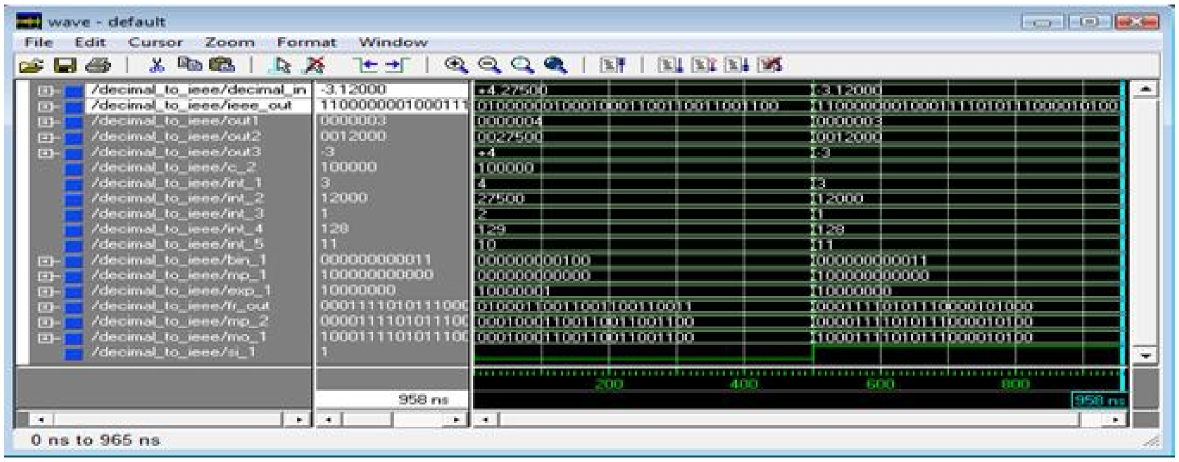
1. Decimal to IEEE-754 : (Single precision) Conversion: Input1

Decimal in = +4.27500
Required IEEE-754 out = 01000000100010001100110011001100
Obtained IEEE-754 out = 01000000100010001100110011001100



```
/decimal_to_ieee/decimal_in   +4.27500              +4.27500                            3.12000
/decimal_to_ieee/ieee_out     010000001000100(      01000000100010001100110011001100    11000000010001111010111000010100
/decimal_to_ieee/out1         0000004               0000004                             0000003
/decimal_to_ieee/out2         0027500               0027500                             0012000
/decimal_to_ieee/out3         +4                    +4                                  3
/decimal_to_ieee/c_2          100000                100000
/decimal_to_ieee/int_1        4                     4                                   3
/decimal_to_ieee/int_2        27500                 27500                               12000
/decimal_to_ieee/int_3        2                     2                                   1
/decimal_to_ieee/int_4        129                   129                                 128
/decimal_to_ieee/int_5        10                    10                                  11
/decimal_to_ieee/bin_1        000000000100          000000000100                        000000000011
/decimal_to_ieee/mp_1         000000000000          000000000000                        100000000000
/decimal_to_ieee/exp_1        10000001              10000001                            10000000
/decimal_to_ieee/fr_out       010001100110011(      01000110011001100110011             00011110101110000101000
/decimal_to_ieee/mp_2         000100011001100       00010001100110011001100             00001111010111000010100
/decimal_to_ieee/mo_1         000100011001100       00010001100110011001100             10001111010111000010100
/decimal_to_ieee/si_1         0
                              465 ns                200   400   600   800
                                                    465 ns
0 ns to 965 ns
```

Input2 : Decimal in = -3.12000
Required IEEE 754 out = 11000000010001111010111000010100
Obtained IEEE 754 out = 11000000010001111010111000010100



```
/decimal_to_ieee/decimal_in   -3.12000              +4.27500                            -3.12000
/decimal_to_ieee/ieee_out     110000000100011       01000000100010001100110011001100    11000000010001111010111000010100
/decimal_to_ieee/out1         0000003               0000004                             0000003
/decimal_to_ieee/out2         0012000               0027500                             0012000
/decimal_to_ieee/out3         -3                    +4                                  -3
/decimal_to_ieee/c_2          100000                100000
/decimal_to_ieee/int_1        3                     4                                   3
/decimal_to_ieee/int_2        12000                 27500                               12000
/decimal_to_ieee/int_3        1                     2                                   1
/decimal_to_ieee/int_4        128                   129                                 128
/decimal_to_ieee/int_5        11                    10                                  11
/decimal_to_ieee/bin_1        000000000011          000000000100                        000000000011
/decimal_to_ieee/mp_1         100000000000          000000000000                        100000000000
/decimal_to_ieee/exp_1        10000000              10000001                            10000000
/decimal_to_ieee/fr_out       000111101011100(      01000110011001100110011             00011110101110000101000
/decimal_to_ieee/mp_2         000011110101110(      00010001100110011001100             00001111010111000010100
/decimal_to_ieee/mo_1         100011110101110(      00010001100110011001100             10001111010111000010100
/decimal_to_ieee/si_1         1
                              958 ns                200   400   600   800
                                                    958 ns
0 ns to 965 ns
```

Device utilization summary: Binary to Decimal for Vertex 2Pro

(Estimated values) Selected Device: 2vp2fg256-6

Logic Utilization	Used	Available	Utilization
Number of bonded IOBs	40	140	28 %

Decimal to Binary for Vertex 4 (Estimated values)

Selected Device: 4vfx12sf363-11

Logic Utilization	Used	Available	Utilization
Number of Slices	276	6144	4 %
Number of 4 inputs LUTs	497	12288	4 %
Number of bonded IOBs	40	240	16 %

Decimal to IEEE-754 for Vertex 4 (Estimated values)

Selected Device: 4vfx12sf363-11

Logic Utilization	Used	Available	Utilization
Number of Slices	4900	6144	79 %
Number of Slice Flip Flops	37	12288	0 %
Number of 4 inputs LUTs	7462	12288	60 %
Number of bonded IOBs	96	240	40 %
Number of GCLKs	1	32	3 %
Number of DSP48s	18	32	56%

Conclusions

This paper deals with development of an n-bit binary to decimal conversion, decimal to n bit binary conversion and decimal to IEEE-754 conversion for floating point arithmetic logic unit (FPALU) using the help of ModelSim (VHDL) and synthesized with Xilinx tools. We have designed n-bit binary to decimal conversion, decimal to n bit binary conversion VHDL programs. All programs are on an n-bit format; hence they can be treated as *Universal programs*. Normally most of the industries now-a-days are using either 4-bit ALU or 32-bit ALU, we have generalized it. Hence now we need not to worry about the bit size of ALU. This will solve all the problems of 4-bit, 8-bit and 16-bit ALU's and so on. Simulation results of all the designed programs have been carried out for various inputs with the help of ModelSim tool.

The code we developed to design this converter, conceptually contains something different and interesting. Floating-point units (FPU) can be implemented in various ways, and the architecture of an FPU has a great affect on its overall performance, area and power dissipation. Arithmetic processor [10], is one of the major fields where we can apply this floating point unit. There we call it floating point processor which does computations in accordance with the IEEE-754 floating-point standard. Also hardware implementation of this arithmetic unit is supposed to be done and it is expected that this will significantly accelerate a wide variety of applications. further it can be optimized to achieve a faster speed or occupy a smaller area for the future. The issue encountered in the VLSI architectures for finite impulse response (FIR) filter is the increased number of components, especially delay elements. For the VLSI architecture reconfigured with reduced register usage, this article provides the floating point processing element (FPPE) implementation with Cross-Folded Shifting.

The proposed FIR filter system reduces the number of components in the circuit which increases the complexity and high delay rate in the logical operation. The system has a comparatively reduced delay rate and power consumption. Hence, an efficient fast architecture based on the FPPE method is developed in this paper.

Acknowledgement

Every success stands as a testimony not only to the hardship but also to hearts behind it. Likewise, the present work has been undertaken and completed with direct and indirect help from many people, my friends, My wife ,My elder daughter Dr. Vaishnavi Sharma (MBBS) and I would like to acknowledge all of them for the same.

Tools Used: Platform: Windows XP, VISTA
Simulator: Model Sim SE, Version 5.5f and 5.5a, Mentor Graphics Company
Synthesizer: Leonardo Spectrum 2003a.33, Mentor Graphics, Inc., Xilinx 6.3i

References

1. "IEEE Standard for Floating-Point Arithmetic", in IEEE Std 754-2008, Vol., no., PP.1-70, Aug. 29 (2008).
2. Jain, Jenil, and Rahul Agrawal. "Design And Development of Efficient Reversible Floating Point Arithmetic unit." In Communication Systems and Network Technologies (CSNT), 2015 Fifth international Conference on, PP. 811-815. IEEE, (2015).
3. Gopal, Lenin, Mohd Mahayadin, Nor Syahira, Adib Kabir Chowdhury, Alpha Agape Gopalai, and Ashutosh Kumar Singh. "Design and synthesis of reversible arithmetic and Logic Unit (ALU)." In Computer, Communications, and Control Technology (I4CT), 2014 International Conference on, PP. 289-293. IEEE, (2014).
4. 754-2019 - IEEE Standard for Floating-Point Arithmetic IEEE Std 754-2019 (Revision of IEEE 754-2008) Published: (2019).
5. Nachtigal, Michael, Himanshu Thapliyal, and Nagarajan Ranganathan. "Design of a reversible floating-point adder architecture." In Nanotechnology (IEEE-NANO), 2011 11th IEEE Conference on, PP. 451-456. IEEE, (2011).
6. Kahan, William. "IEEE standard 754 for binary floating-point arithmetic."Lecture Notes on the Stat us of IEEE 754.94720-1776 11 (1996).
7. IEEE standard 754-2008. IEEE standard for floating-point arithmetic. IEEE Computer Society, Aug (2008).
8. E. M. Schwarz, J. S. Kapernick, and M. F. Cowlishaw. Decimal Floating-Point Support on the IBM System z10 Processor. *J. IBM Res. and Dev.*, Jan (2009).
9. Xilinx Inc. Xilinx University Program Virtex-II Pro Development System, Hardware Reference Manual.
10. IEEE Computer Society (2019-07-22). IEEE Standard for Floating-Point Arithmetic. IEEE STD 754-2019. IEEE. PP. 1–84. doi:10.1109/IEEESTD.2019.8766229. ISBN 978-1-5044-5924-2. IEEE Std 754-2019.
11. ISO/IEC 60559:2020 Information technology Microprocessor Systems Floating-Point arithmetic. Iso.org. May 2020. PP. 1–74.