SIMULATION AND SYNTHESIS OF 32-BIT MULTIPLIER USING CONFIGURABLE DEVICES

Dinesh Kumar[1] and Girish Chander Lall[2]
ECE Deptt., MMEC, Mullana, India
ECE Deptt., HCTM, Kaithal, India

ABSTRACT

Floating- point numbers are frequently used for numerical calculations in computing systems for better accuracy, but floating- point operations are complex and difficult to design on FPGAs . This work attempts to design such hardware architecture for single precision floating-point multiplication that is easily implementable with high efficiency. The multiplier unit is based on ancient vedic mathematics technique. The proposed design is described using VHDL which is simulated using Modelsim SE 5.7f and synthesized using ISE Xilinx 10.1i on FPGA device Virtex -XC4VSX25-12FF668.Logic utilization shows that the utilization of slices is 1% and of 4-input LUTs is 21%.Alsologic distribution indicate that number of occupied slices are 2358 which are fully related.

KEYWORDS: *Multiplier, Single Precision, Reconfigurable, Floating point, FPGA*

I. INTRODUCTION

The digital signal processing landscape has long been dominated by microprocessors with enhancements such as single cycle multiply-accumulate instructions and special addressing modes. Various commercial hardware products of signal processing algorithms have been widely utilized in many applications. The manufacturers always try to find a high- performance and low- cost solution in order to make products marketable and profitable.

There are many data processing applications (e.g., image and voice processing), which use a large range of values and need a relatively high precision. In such cases, instead of encoding the information in the form of integers or fixed-point numbers, an alternative solution is a floating-point representation [1,2]. Floating point operations are hard to implement on FPGAs because of the complexity of their algorithms. They usually require excessive chip area, a resource that is always limited in FPGAs. This problem becomes even harder if 32-bit floating point operations are required. On the other hand, many scientific problems require floating point arithmetic with high levels of accuracy in their calculations. Furthermore, many of these problems have a high degree of regularity that makes them good candidates for hardware accelerated implementations. Thus, the necessity for 32-bit floating point operators implemented in FPGAs arises. In an effort to accommodate this need, we use the IEEE 754 standard for binary floating point arithmetic (single precision) to implement a floating-point multiplier [3]. The reason for choosing multiplier for this study is that the multiplication is an essential mathematical operation in computing systems which require more number of steps for the processing and the design. The multiplier is built using vedic multiplier [4] for the significant (mantissa) multiplication to obtain a faster implementation. The design details and the performance based on speed and area requirements, of these operators are discussed in this paper.

This paper is organized in the following way: Section II highlights the related work. In Section III a brief review of IEEE 754 standard for binary floating-point arithmetic is given. In the next section, the multiplier is described in detail. Then in section V, the synthesis results are demonstrated. Finally, in section VI the paper is concluded with the applications of the operators and prospects for future improvements in these implementations.

II. RELATED WORK

Similar work was presented [5] for single precision floating-point multiplication in which they used digit-serial arithmetic which is not so fast method. Also, proper rounding techniques were not implemented. However, they presented a pipelined structure in order to produce a result every clock cycle. A work for floating-point division is reported [6] for embedded VLSI integer processors with no hardware unit. They focused on high-radix digit-recurrence algorithms. In these implementations, 32 bit operators are designed. However, by using a small floating point format (16 bits or 18 bits wide), smaller and faster implementations can be built but with less accuracy.
There are number of problems associated with tree and array multipliers [7]. Tree multipliers have shortest logic delay but irregular layouts with complicated interconnects. Irregular layouts introduce significant interconnect delay and make noise a problem and the delay of the interconnection is not suitable for VLSI implementation. Similarly array multipliers have larger delay and significant amount of power is consumed.

III. FLOATING POINT FORMAT

The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most widely-used standard for floating-point computation, and is followed by many CPU and FPU implementations. The standard defines formats for representing floating-point numbers (including negative zero and denormal numbers) and special values (infinities and NaNs) together with a set of floating-point operations that operate on these values. It also specifies four rounding modes and five exceptions (including when the exceptions occur, and what happens when they do occur). Four formats for representing floating-point values are: single-precision (32-bit), double-precision (64-bit), single-extended precision (≥ 43-bit, not commonly used) and double-extended precision (≥ 79-bit, usually implemented with 80 bits). The basic format for single precision is further divided into sign, exponent, and mantissa part as shown in Fig 3.1.

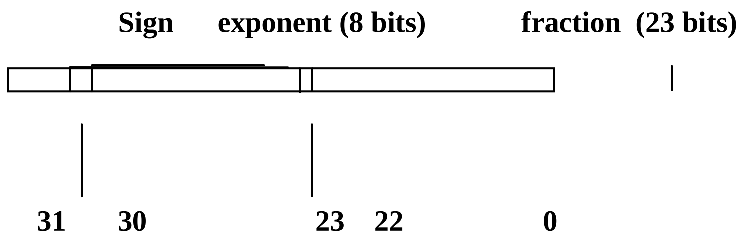


Fig-1: Format for Single-Precision

Biased Exponent: A constant is added to the actual exponent so that the biased exponent is always a positive number. The 8-bit exponent for single-precision floating-point numbers can take any value in the range -126 to +127.

3.1 Normalized Numbers

In most cases, the floating-point real numbers are represented in normalized form. A normalized real number consists of a normalized significand that represents a real number between 1 and 2 and an exponent that specifies the number's binary point. The requirement that the leftmost digit of the significand be nonzero is called normalization.

3.2 Real Numbers and non number Encodings

A variety of real numbers and special values can be encoded in floating-point format. These numbers and values are generally divided into the following classes:

1. Signed Zeros

Zero can be represented as a +0 or a -0 depending on the sign bit. Both encodings are equal in value. The sign of a zero result depends on the operation being performed and the rounding mode being used.

2. Normalized and Denormalized Finite Numbers

Non-zero, finite numbers are divided into two classes: normalized and denormalized. The normalized finite numbers comprise all the non-zero finite values that can be encoded in a normalized real number format between zero and infinity (∞). The denormalized number is computed through a technique called gradual underflow.

3. Signed Infinities

The two infinities, +∞ and -∞, represent the maximum positive and negative real numbers, respectively, that can be represented in the floating-point format. Infinity is always represented by a zero significand (fraction and integer bit) and the maximum biased exponent allowed in the specified format

4. Not a Number (NaN)

There are two classes of NaN: quiet NaNs (QNaNs) and signaling NaNs (SNaNs). A QNaN is a NaN with the most significant fraction bit set; a SNaN is a NaN with the most significant fraction bit clear.

5. Indefinite

For each FPU data type, one unique encoding is reserved for representing the special value indefinite.

3.3 Conversion and Rounding

When a number is represented in some other format (such as a string of digits), then it will require a conversion to be used in floating-point format. When the result is obtained from an operation, it may not be possible to represent exactly in the IEEE 754 standard. Therefore, rounding is needed before the result is stored in the memory or registers, and/or sent to the output. In round to nearest even the value is rounded up or down to the nearest infinitely precise result. In round up and down the number will be rounded up towards +∞ and -∞ respectively. Round towards zero modes is not used in general.

3.4 Exceptions

There are four types of exceptions that should be signaled through a one bit status flag when encountered. Some arithmetic operations are invalid, such as a division by zero or square root of a negative number. The result of an invalid operation shall be a NaN.
Inexact exception should be signaled whenever the result of an arithmetic operation is not exact due to the restricted exponent and/or precision range. Two events cause the underflow exception to be signaled, tininess and loss of accuracy. The overflow exception is signaled whenever the result exceeds the maximum value that can be represented due to the restricted exponent range. It is not signaled when one of the operands is infinity, because infinity arithmetic is always exact.

IV. FLOATING POINT MULTIPLIER

Multiplication is one of the operations which require more number of steps in computation [8]. However, floating point multiplication is somewhat simpler than addition to implement because the significands are represented in sign- magnitude format, which is similar to the integer format. The only additional step required is the calculation of the correct exponent. However, several special cases must be considered. First, if the product is 0, the exponent must be set to the largest negative value which, in the biased case, is 0. Second, if the resulting exponent is too large in magnitude, an exponent overflow is said to have occurred, which cannot be corrected, hence, an overflow indicator must be tuned on. Finally, denormalized numbers and NaNs must be given special attention. An algorithm to implement floating-point multiplication must take into account all of these possibilities. The algorithm used here is explained in the following subsection.

4.1 Multiplication Algorithm

As explained in previous section, a binary floating-point number is represented by a sign bit, the significand and the exponent. Given two numbers Operand A and Operand B, the flowchart in figure 2 can be used to compute their product, given that e_A, e_B and $frac_A$, $frac_B$ are the exponents and significands of the numbers, respectively. A detailed description of the algorithm follows:

1. The hidden bit (24th bit) is made explicit. If $^e a$ or $^e b = 0$, it is made ‘0’, otherwise a ‘1’. At this point 33 bits are needed to store the number, 8 for the exponent, 24 for the significand and 1 for the sign.

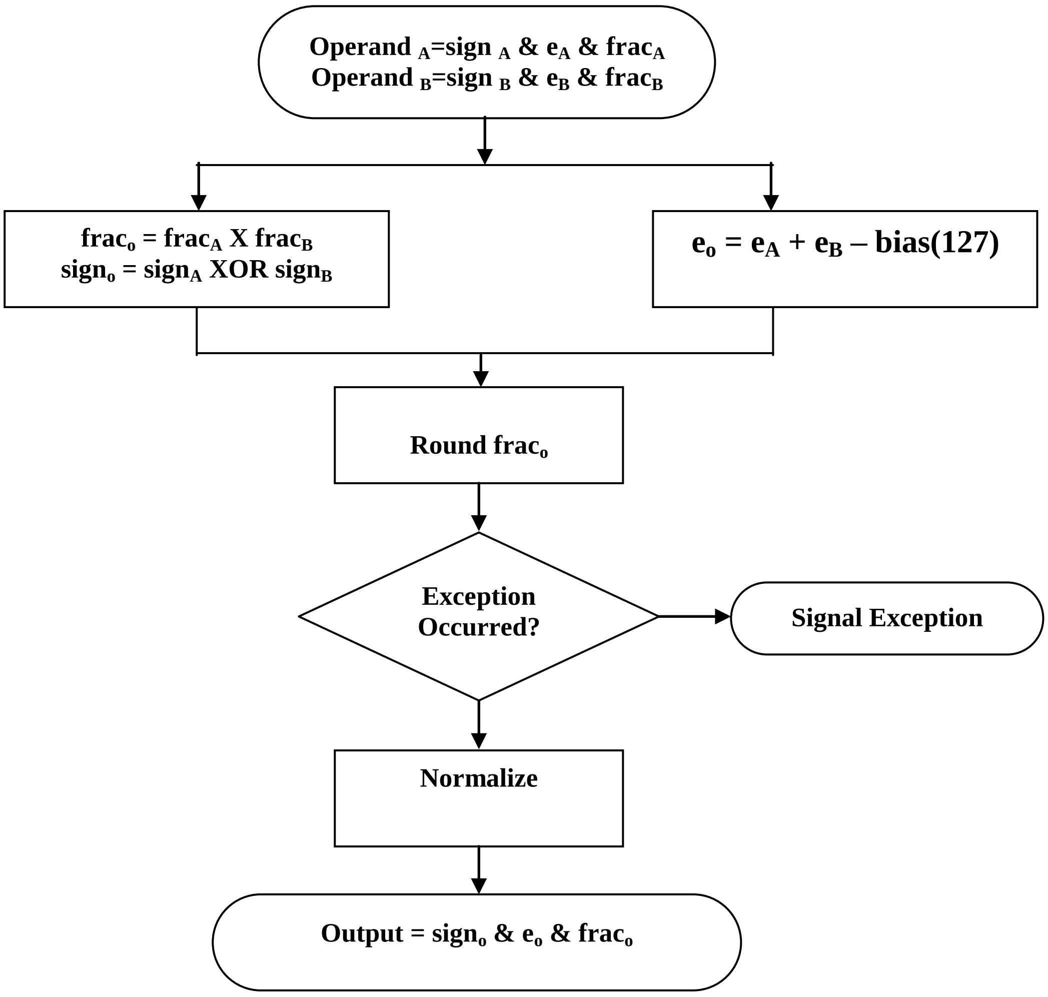

Operand A=sign A & eA & fracA
Operand B=sign B & eB & fracB
frac_o = fracA X fracB
sign_o = signA XOR signB
e_o = eA + eB – bias(127)
Round frac_o
Exception Occurred?
Signal Exception
Normalize
Output = sign_o & e_o & frac_o

Fig-2: Floating-point Multiplication Algorithm

2. The result of the multiplication is given by the formula:

Sign =$sign_A$ xor $sign_B$, $e = e_A + e_B$,

Frac = $frac_A$ x $frac_B$

The addition of the exponents is a trivial operation as long as we keep in mind that they are biased. This means that in order to get the right result, we have to subtract **127** (bias) from their sum. The sign of the result is just the XOR of the two sign bits. The multiplication of the significands is just an unsigned, integer multiplication. A brief explanation of the method used to perform this multiplication is given in Next Section.

3. The product of two 24-bit numbers can be 48-bit wide. But only, 24 bits can be accommodated for the significand. Therefore, 48-bit result is rounded up to 24 bits. There are four methods for rounding: Round-to-nearest-even, round-up, round-down and round-to-zero; in which round-to-nearest-even is the most widely used.

Since the result precision is not infinite, sometimes rounding is necessary. To increase the precision of the result and to enable round-to-nearest-even rounding mode, three bits were added internally and temporally to the actual fraction: *guard*, *round*, and *sticky* bit. While guard and round bits are normal storage holders, the sticky bit is turned '1' whenever a '1' is shifted out of range.
4. There must be a leading '1' in the significand of any floating-point number (unless it is not denormalized). To make the MSB '1' in the result, the bits are shifted left, and with each shift, the exponent is incremented by 1. This way normalization is done.
5. The result is assembled into the 32 bit format, neglecting the 24th bit of the significand.

4.2 Vedic Multiplier (Urdhav-Triyak method)

The multiplier A[n] is of size 'n' words and the multiplicand B[m] is of size 'm' words, where A and B are given by equation 1 and 2**.**

$$A[n] = \sum_{i=0}^{n-1} a_i * X^i \qquad (1)$$

$$B[m] = \sum_{i=0}^{m-1} b_i * X^i \qquad (2)$$

Product of A and B is given by equation 3.

$$P\,[n + m] = A\,[n] * B\,[m] \qquad (3)$$

$$\sum_{i=1}^{n} CP[0,0,i] * X^{i-1} +$$
$$\sum_{j=1}^{m-n} CP[0,j,n] * X^{n+j-1} +$$
$$\sum_{k=1}^{n-1} CP[k,k+m-n,n-k] * X^{m+k+1}$$

Where

$$CP[n,m,q] = \sum_{i=n}^{n+q-1} a_i * b_j \qquad (4)$$

Equation 4 gives the cross-product of two numbers. Where j = (m+ n + q - i -1)

3.2 Simulation Setup

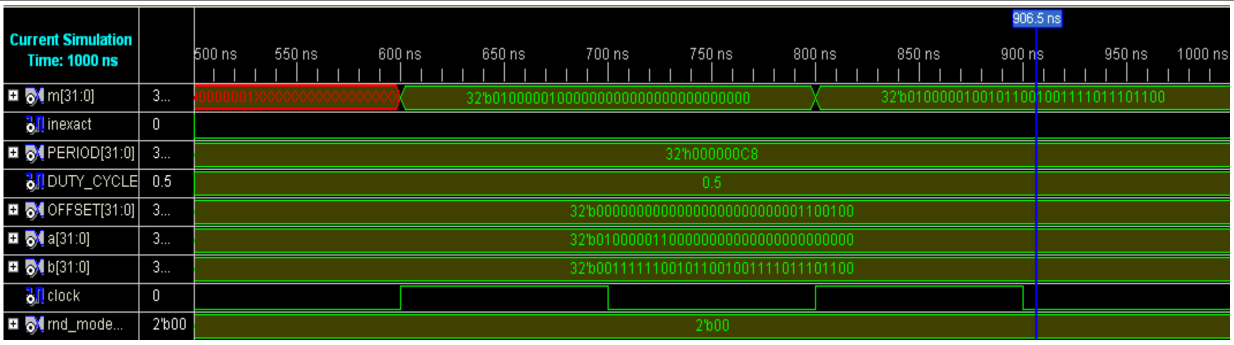
The code is written in Hardware Description Language and synthesized using Xilinx 10.1i .The device used is Virtex **XC4VSX25-12FF668** and logic simulation is done using Model Sim SE5.7f.

V. RESULTS

The synthesis results (design summary) are shown in Table 1. From the table it is clear that number of slices used is 154 out of 20480 with utilization of 1%.Logic distribution shows utilization of occupied slices is 23% and total number of 4inpur LUTs used are 4470 out of 20480.
Timing Summary indicates the clock period is 32.730 ns (frequency: 30.553MHz) and total time taken is 32.730 ns (11.651ns for computation, and 21.079 ns for communication) .
Macro Statistics shows total number of logical component used are : Adders/Subtractors: : 102,16-bit adder:8, 24-bit adder :1, 6-bit adder :70,8-bit adder:10, 8-bit subtractor: 1, 9-bit adder :10,9-bit subtractor:2,Registers:123,FlipFlops:123,Comparators:6, Multiplexers : 1, Logic shifters : 2, 24-bit shifter logical left:1,48-bit shifter logical right:1.
Simulation result of FP Multiplier is shown in Fig-3. Power analysis indicates %age utilization is 31.3 as shown in Fig-4 and RTL schematic view is shown in Fig-5.

Table-1 : Device Utilization Summary

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	154	20,480	1%
Number f 4 input LUTs	4,455	20,480	21%
Logic Distribution			
Number of occupied Slices	2,358	10,240	23%
Number of Slices containing only related logic	2,358	2,358	100%
Number of Slices containing unrelated logic	0	2,358	0%
Total Number of 4 input LUTs	**4,470**	**20,480**	**21%**
Number user as logic	4,455		
Number used as a route-thru	15		
Number of bonded IOBs	100	320	31%
Number of BUFG/BUFGCTRLs	1	32	3%
Number user as BUFGS	1		



```
m[31:0]         3...    0000001XXXXXXXXXXXXXXXXX  32'b01000001000000000000000000000000  32'b01000001001011001001111011101100
inexact         0
PERIOD[31:0]    3...    32'h000000C8
DUTY_CYCLE      0.5     0.5
OFFSET[31:0]    3...    32'b00000000000000000000000001100100
a[31:0]         3...    32'b01000001100000000000000000000000
b[31:0]         3...    32'b00111111100101100100111011101100
clock           0
rnd_mode...     2'b00   2'b00
```

Fig- 3: Simulation result of FP Multiplier

Name	Power [W]	Used	Total Available	Utilization [%]
Clocks	0.018	1	….	….
Logic	0.000	4463	20480	21.8
Signals	0.000	4402	….	….
Ios	0.000	100	320	31.3
DCMs	0.000	0	4	0
Total Quiescent Power	0.393			
Total Dynamic Power	0.018			
Total Power	0.412			

Fig-4: Power Analysis by Xilinx Xpower Analyzer

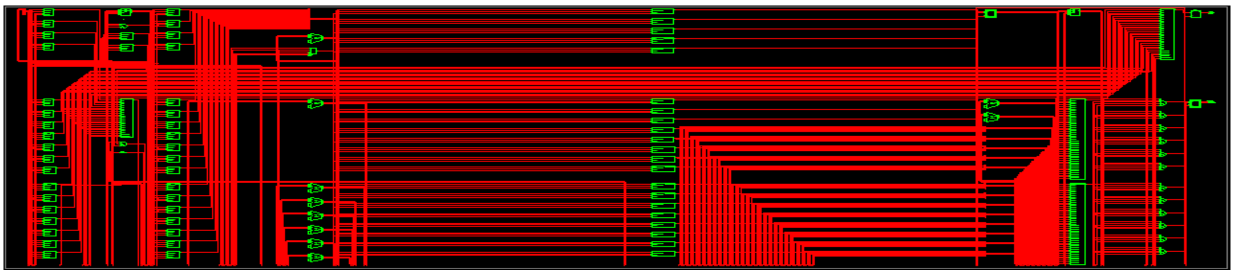


Fig-5: RTL Schematic view

VI. CONCLUSIONS

In this paper, single precision floating point multiplier is designed using ancient vedic mathematics technique. Algorithm is implemented using ISE Xilinx 10.1i on FPGA device Virtex -XC4VSX25-12FF668 and *s*imulation is done by using Modelsim SE 5.7f. The synthesis results shows that number of slices used is 154 out of 20480 with utilization of 1%.Logic distribution shows utilization of occupied slices is 23% and total number of 4inpur LUTs used are 4470 out of 20480.
Timing Summary indicates total time taken for the process is 32.730 ns (11.651ns for computation, and 21.079 ns for communication) and number of component used are also shown.
Double precision multiplier may be designed as extension of this work. Also we can use Nikhlam sutra (another technique of vedic mathematics) and compare the results. We can also design matrix multiplier and compare speed and area of operations.

REFERENCES

[1] John L. Hennessy and David A. Patterson. Computer Architecture A Quantitative Approach, Second Edition. Morgan Kaufmann, 1996.
[2] Deschamps, Jean-Pierrie and Sutter, D. Gustavo, Synthesis of Arithmetic Circuits, FPGA, ASIC and Embedded Systems, John Wiley & sons Inc. Publication (2006).
[3] Perry, Douglas, VHDL Programming by Example, McGraw Hill Publication (2002).
[4] Jagadguru Swami Sri Bharath, Krsna Tirathji, "Vedic Mathematics or Sixteen Simple Sutras from the Vedas", Motilal Banarsidas, Varanasi, India, 1986.
[5] Loucas Louca, Todd A. Cook, William H. Johnson, "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs", IEEE, Sept. 1996.
[6] C.P. Jeannerod, S. Raina and A. Tisserand, " High-Radix Floating-Point Division algorithms for Embedded VLIW Processors", Arenaire Project, 2003.
[7] GH. A. Aty, Aziza 1. Hussein, I. S. Ashour and M. Mona,"Highspeed,Area-Efficient FPGA-Based -Floating-point Multiplier",Proceedings ICM 2003, pp-274-277,Dec. 9-11 2003, Cairo, Egypt.
[8] H. Thapliyal and M. B. Srinivas, "High Speed Efficient N XN Bit Parallel Hierarchical Overlay Multiplier Architecture Based on Ancient Indian Vedic Mathematics", Enformatika Trans., vol. 2, pp. 225-228, December 2004.
[9] Nabeel Shirazi, A1 Walters, and Peter Athanas."Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines". In IEEE Symposium on FPGAs **for** Custom Computing Machines, pages 155-162, April 1995.
[10] S.E. Cquillan, J.V. McCanny, and R. Hamill, "New Algorithms and VLSI Architectures for SRT Division and Square Root," Proc. 1I h symp. Computer arithmetic, pp. 80-86, Windsor, Ontario, Canada, June 29-jully 2, 1993.
[11] M. E. Louie and M. D. Ercegovac," On Digit -Recurrence Division Implementations for Field Programmable gate Arrays" In Proc. Of the 1 10h symposium on Computer Arithmetic, PP. 202-209, Canada, June29 -July 2 1993
[12] T. Callaway and E. Swartzlander. Optimizing multipliers for WSI. In Fifth Annual IEEE International Conference on Wafer Scale Integration, pages 85-94, 1993.
[13] S. Krithivasan and M. Schulte, .Multiplier architectures for media processing,. in Proc. IEEE 37th Asylomar Conference on Signals Systems, and Computers,, Paci_c Grove, CA, USA, Nov. 2003, pp. 2193.2197.
[14] Shimada and Akinori Kanasugi, " A Dynamically Reconfigurable Arithmetic Circuit for Complex Number and Double Precision Number" World Academy of Science, Engineering and Technology 54 2009.
[15] Umer Nisar Misgar, Wasim Ahmad Khan, and Najeeb-ud-din, " Design of a Floating Point Fast Multiplier with Mode Enabled" in proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Vol II IMECS 2009, March 18 - 20, 2009, Hong Kong
[16] Pardeep Sharma, Ajay Pal Singh,"Implementation of Floating Point Multiplier on Reconfigurable Hardware and Study its Effect on 4 input LUT's", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 7, July 2012, pp 244-248.

AUTHORS

Dinesh Kumar is working as Associate Professor in Department of Electronics and Communication Engineering department of M. M. Engineering College, Mullana with experience of 17 years. He has guided twelve MTech students and published about 25 papers in national and international conference and journals. He has completed M tech.in 2007 from

GZSCET, Bathinda and currently pursing PhD.

G. C. Lall is working as Professor in Department of Electronics and Communication Engineering department of HCTM Kaithal with experience of 45years.He has guided forty MTech students and published about 15 papers in national and international conference and journals.