International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 01 | Jan 2020 www.irjet.net p-ISSN: 2395-0072

IMPLEMENTATION OF FLOATING POINT FFT PROCESSOR WITH SINGLE PRECISION FOR REDUCTION IN POWER

R. Balasaraswathi[1], D. Divya[2], M. Harinikalayani[3], I. Vivek Anand M.E[4], and Dr. T.S. Arun Samuel[5]

[1,2,3]Student, Department of ECE, National Engineering College, Kovilpatti, India
[4,5]Department of ECE, National Engineering College, Kovilpatti, India

---

Abstract - *The advance technology of VLSI has been the enhancing special feature in the appearance of VLSI circuits that can handle floating point (FP) arithmetic. Depending on the various processor applications requirements also differ, i.e. some processors have a high repertoire of functions but results in low performance, while some processors aim at achieving the highest throughput that leads to use more operations such as multiply and add and that can produces more latency. For real-time processing requirements, performing a large amount of FP operations are considered as a major bottleneck due to the excessively long run time required. In many cases FP arithmetic requires additional operations such as alignment, normalization and rounding, giving rise to some significant increase in terms of area, power consumption and computational latency .Such a problem might be mitigated by employing the fused FP add-subtract and dot-product units specially designed to perform those tedious tasks.*

For achieving high performance with minimizing hardware complexities, existing rounding algorithms like mantissa, exponent and sign are used to generate two consecutive values in parallel, and compute the rounded product by using these values. This research work focuses on reducing computation time, area and the power compared to many existing floating point adder consumption by developing a new floating-point architecture. Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain representation and vice versa. The FFT processor architecture exploits the superior area utilization efficiency existing with the single-path delay feedback (SDF) in memory and the single-path delay commutate (SDC) in adder. The circuits are designed by Encounter RTL (digital design) using Cadence and the simulation results will be observed using cadence tool.

Key Words: Floating-point, ALU, Pipelining, Precision

1. INTRODUCTION

In domain of digital signal processing the number representation in the form of fixed-point or floating-point[1].The contribution deals with a binary representation of real-numbers. The advantage of *floating-point* representation over fixed-*point* representation that can support much a wider range of values in integer representation. The representation of real numbers in Floating Point Unit (FPU) typically used binary *floating-point* format numbers [2] which is used to increase the speed and efficiency compared to fixed-point representation. For achieving accuracy and efficiency in digital and radar imaging and to reduce the complexities during the processing, floating-point representation played a major role.

Floating-point unit designed for applications such as space craft, launching rockets and big data. Since integer arithmetic lacks the range and precision for the accuracy, *VLSI technology* making *it to be* possible. There are many processors with fixed or floating-point representation and there are also several blocks used for arithmetical operations. In high resolution radar imaging applications for performing the task of pulse compression, Floating-Point (FP) Fast Fourier Transform (FFT) processors are often used.

2. FLOATING POINT

A system which describes the representing numbers that would be too large or too small as integers is called as floating point number. Compared to fixed point representation, floating point representation is able to retain its resolution and accuracy[3]. The sign, mantissa and exponent can make a floating-point number which shown in Fig 1.*S* is the Sign bit (0 is positive and 1 is negative).The sign bit is represented either as sign or magnitude. *E* is the exponent bit, very large numbers have large positive exponent and Very small close-to-zero numbers have negative exponents. The range of values is increased in exponent field. *M* is the Fraction bit or Mantissa (fraction after binary point). The precision of FP numbers can be improve by having More bits in fraction field.

SIGN	EXPONENT	MANTISSA

Fig. 1. Representation of Floating Point

In 1985, Institute of Electrical and Electronics Engineers (IEEE) established a technical standard for floating point arithmetic (IEEE standard 754) [4]. The IEEE 754 standard addressed many problems found in the diverse floating- point implementations that made them difficult to use reliably and portably. Many hardware floating-point units use the standard.

In accordance with IEEE standard 754, Conversion of decimal to the floating point consists of three steps such as

conversion of the decimal to binary then second step, is the converted binary is represented in scientific notation. Scientific representation is done by shifting the point towards left in converted binary number and multiplied by exponent i.e., multiplied by 2^n (where n is number of shifts) Then the third step is to convert scientific notation to floating point according to standard 754. The sum of exponent bias and n can make the exponent bits. The exponent bias is constant which were denoted as 127 for single precision and 1023 for double precision.

The smallest change that occurred in floating point representation is known as precision. The meaning of precision implies 'closeness' or 'accuracy'. There are two types of precision; they are single precision and double precision [5].

The standard representation of Single precision floating point consists of 32 bits, which may be represented as numbered from 0 to 31, left to right as shown in Figure 2.

Sign (1)	Exponent(8	Mantissa(23)

Fig. 2. Representation of Single Precision

The standard representation double precision floating point requires a 64 bits, which may be represented as numbered from 0 to 63, left to right as shown in figure3.

Sign (1)	Exponent (11)	Mantissa (52)

Fig. 3 Representation of Double Precision

ADDITION OF FLOATING POINT NUMBER

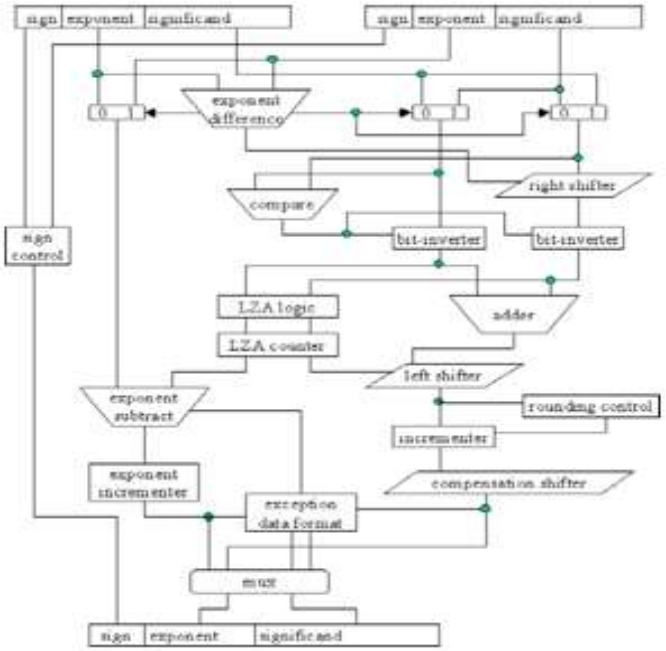


Fig. 4. Architecture of floating point addition

The architecture of floating point addition shown in Figure 4,Which consists blocks such comparator ,sign control, bit inverter, adder, LZA logic which replaces the carry look ahead adder and used to speed up the process ,counter ,left shifter ,right shifter, exponent incrementer, Exponent subtract, incrementer , rounding control, exception data format and multiplexer which replaces the encoder for better results. These all blocks help to achieve increase in efficiency and reduce to the power consumption, area consumption and minimum power delay.

Normalization of floating point is desirable, i.e., there will be only one significant digit (binary can only be 1) which is to the left of the radix point of mantissa m. The addition of floating-point numbers involved in normalization of floating-point addition, and achieved a normalized sum S. M1 and M2 are designated as the mantissas of two floating- point numbers respectively, and E1and E2 are the designated as exponents [6] .

In alternative floating-point formats to IEEE 754, mantissas M1 and M2 can be represented in 2's complement format to perform addition or subtraction, in a single simple structure. There are three general steps for Summing addends A and B .They are: 1) de- normalization of the addends, 2) addition of mantissa, and 3) normalization of the Sum. If exponents E1 and E2 are not equal during de-normalization of the addends, then the addends must be de-normalized until E1 and E2 match. In typical method addends should be de-normalized to increase the smallest exponent, by X which is equal to the largest exponent E, and by shifting the binary point of mantissa M bits of the addend with the smallest exponent X places to the left which leads to achieve the de-B), the above method described that it would increase powers, by 6 So that powers, equals E, (in this case, E, or 10). If mantissa sum Ms., is not in normalized form, then it is normalized to yield normalized sum S. In other words, if required, the binary point of mantissa sum., is shifted left or right until there is only one significant digit to the left of the binary point to achieve normalized mantissa sum. The normalized sum S can be achieved by adjusting $E_{largest}$ to yield the exponent Es.

A floating-point operation yields a result which is not be represented in the floating-point numbering system used and then an exception occurs[7].There are three types of exceptions, they are Overflow, Underflow, and Zero. After the addition of floating point numbers there would be a chance of occurring overflow or underflow i.e., the absolute value of the result would be either too small(underflow) or too large(overflow) .In accordance with IEEE 754 32-bit single-precision format, which is not capable of representing a positive number greater than ranges from 2^0x2^{127} to $2^{-23}x2^{127}$(positive overflow) or less than 2^{-126} (positive underflow), or a negative number the absolute value of which is greater than from 2^0x2^{127} to $2^{-23}x2^{127}$ (negative overflow), or less than 2^{-126}(negative underflow). According to IEEE 754, which implied that leading digit of 1 so that it is incapable of naturally representing 0 (zero exception).

DESIGN OF FLOATING POINT ADDITION

In Accordance to the architecture of floating point addition may have many blocks which can be individually designed using Modelsim simulator tool and get correlated to form the overall architecture. This architecture can be build by step by step Pipelining process, following steps are:

The first step is that the exponents of two floating point numbers got compared and the absolute value of difference between the two exponents can be calculated. Then larger exponent was considered as the tentative exponent of the result[8].

Then the second step is to make the adjustments in exponent part i.e., to shift the significant of the number with the smaller exponent to right through a number of bit positions that is equal to the exponent difference. The guard (G) and Round (R) bits are two of the shifted out bits of the aligned significant [9].

Thus the effective width of aligned significant must be p + 2 bits for p significant bits. Then the third bit is to be appended which named as sticky bit (S), at the right end of the aligned significant. The sticky bit is the logical OR operation of all shifted bits.

The third step is to add the mantissa part. Let the result of this is SUM.

The fourth step is, during addition of mantissa part, check SUM for carry out (C_{out}) from the MSB position. If a carry out is detected then SUM was shifted right by one bit position and increment the final exponent by 1[10].

Then the final step is that the exception conditions were evaluated, if any. If the logical condition R"(M0 + S") is true then the result get rounded, where M0 represents p^{th} and R" represents $(p + 1)^{st}$ bits from the left end of the normalized significant. S" represents the new sticky bit which occurred by doing the logical OR operation of all bits towards the right of the R" bit. If the rounding condition is true, a 1 is added at the pth bit (from the left side) of the normalized significant. Rounding can be generated a carry-out, as the step 4 has to be performed again when p MSBs of the normalized significant are 1's.

RESULT AND DISSCUSION COMPARATOR

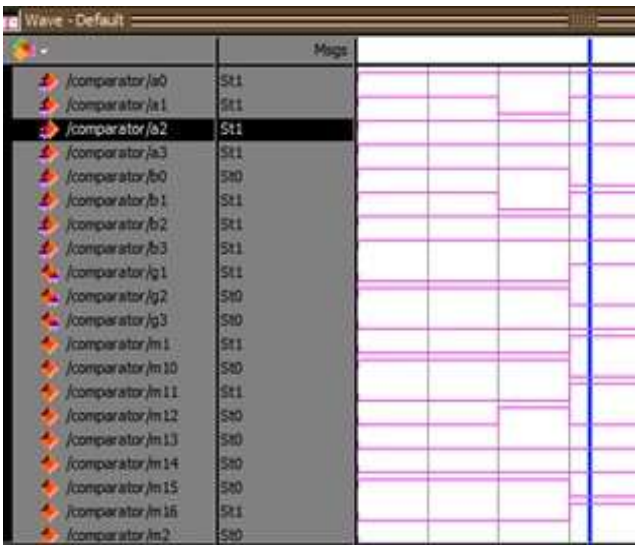


Fig. 5 Simulation result of 4-Bit comparator for a>b condition

As shown Figure 5, a0,a1,a2,a3 and b0,b1,b2,b3 are the inputs. Then a=15 in terms binary a0 a1 a2 a3=1111 and b=7 in terms of binary b0 b1 b2 b3=0111. The result is g1=1.

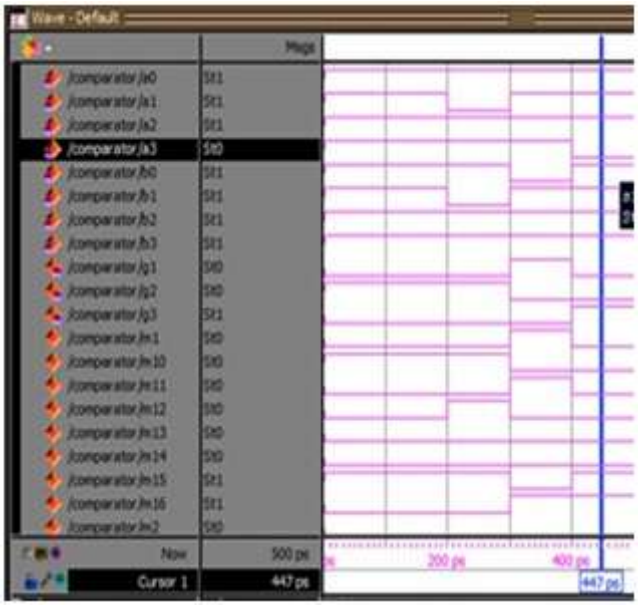


Fig. 6 Simulation result of 4-Bit comparator for a<b condition

As shown in Figure 6, a0 a1 a2 a3 and b0 b1 b2 b3 are the two 4 bit inputs.a0 a1 a2 a3=1110, b0 b1 b2 b3 =1111.The output is g3=1.

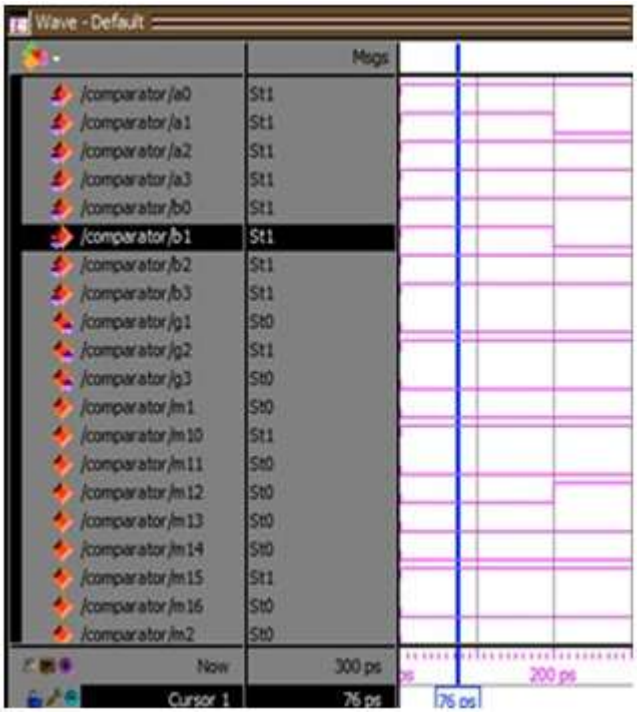


Fig. 7 Simulation result of 4-Bit comparator for a=b condition

As shown +in figure 7, a0 a1 a2 a3 and b0 b1 b2 b3 are the two 4 bit inputs.a0 a1 a2 a3=1111, b0 b1 b2 b3 =1111.The output is g2=1.

SHIFTER

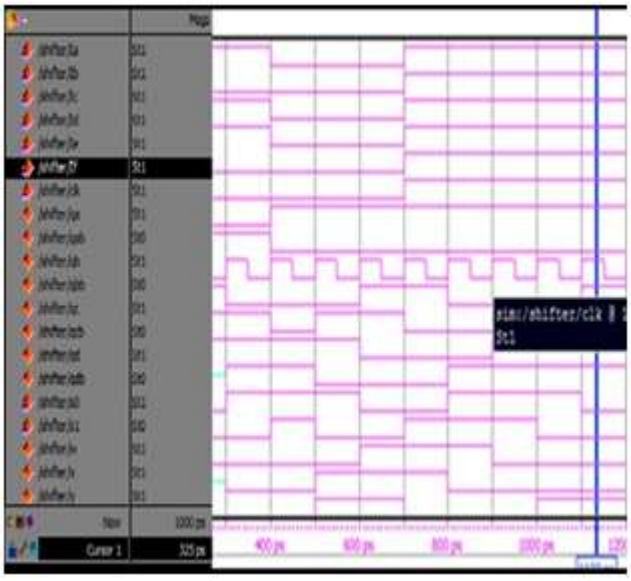


Fig. 8 Simulation result of Right shifter

As shown in figure 8, the inputs are Ia Ib Ic Id Ie If and s0 and s1 are the select lines. Ia Ib Ic Id Ie If=111111 and then s0=1 and s1=0 then the output is 011111.

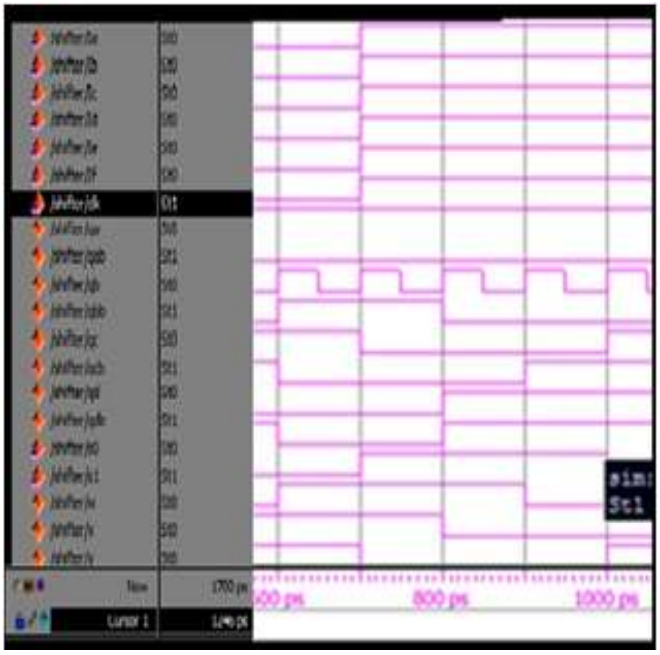


Fig. 9 Simulation result of Left shifter

As shown in Fig 9 the inputs are Ia Ib Ic Id Ie If and s0 and s1 are the select lines. Ia Ib Ic Id Ie If=111111 and then s0=0 and s1=1 then the output is 111110.

EXPONENT INCREMENTER

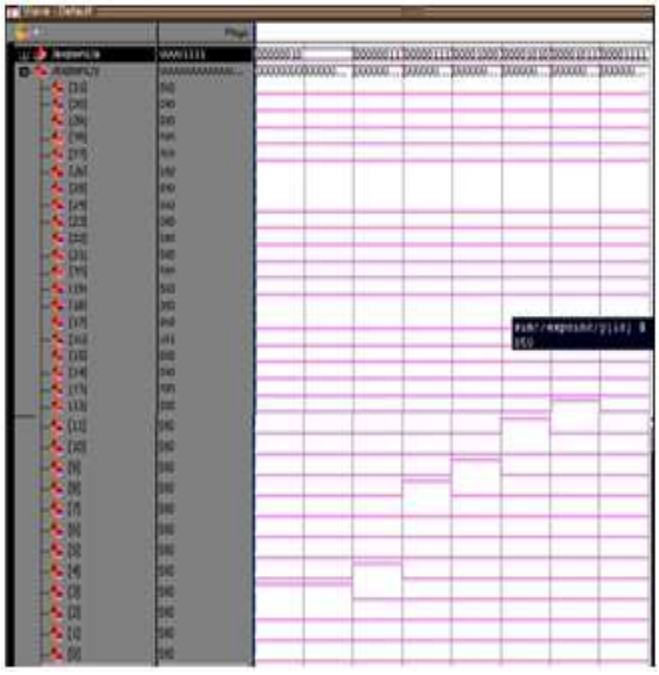


Fig.10 Simulated result of exponent incrementer

As shown in Fig 10, a7 a6 a5 a4 a3 a2 a1 a0 be the input a7 a6 a5 a4 a3 a2 a1 a0=00000100.The result is y=00001000.

EXPONENT DECREMENTER

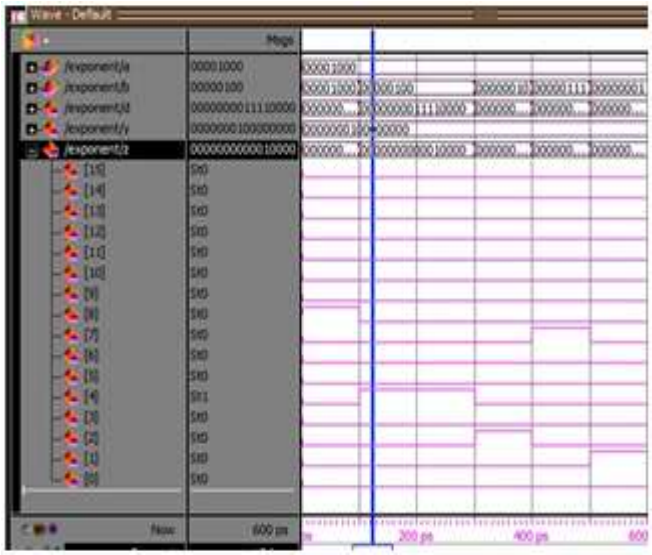


Fig. 11 Simulation result of Exponent decrementer

As shown in Fig 11, a7 a6 a5 a4 a3 a2 a1 a0 and b7 b6 b5 b4 b3 b2 b1 b0 be the input a7 a6 a5 a4 a3 a2 a1 a0=00001000, b7 b6 b5 b4 b3 b2 b1 b0=00000100.The result is y=0000000000010000.

FLOATING POINT ADDITION

Fig.12 Simulation result of floating point addition

As shown Fig 12, the simulation result of floating point addition with single precision using Modelsim Quartus-2.

The following parameters were measured using cadence tool.

The following outputs are obtained from the cadence:

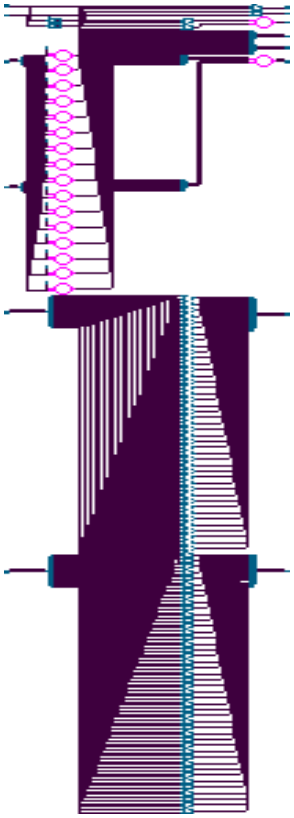
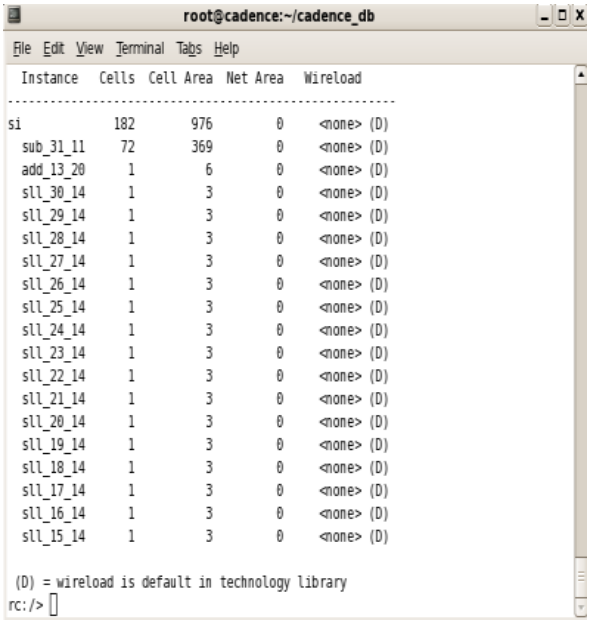


Fig.13 RTL View of Floating Point Addition with Single Precision



```
root@cadence:~/cadence_db
File  Edit  View  Terminal  Tabs  Help
  Instance   Cells  Cell Area  Net Area   Wireload
-----------------------------------------------------------
si             182        976         0    <none> (D)
  sub_31_11     72        369         0    <none> (D)
  add_13_20      1          6         0    <none> (D)
  sll_30_14      1          3         0    <none> (D)
  sll_29_14      1          3         0    <none> (D)
  sll_28_14      1          3         0    <none> (D)
  sll_27_14      1          3         0    <none> (D)
  sll_26_14      1          3         0    <none> (D)
  sll_25_14      1          3         0    <none> (D)
  sll_24_14      1          3         0    <none> (D)
  sll_23_14      1          3         0    <none> (D)
  sll_22_14      1          3         0    <none> (D)
  sll_21_14      1          3         0    <none> (D)
  sll_20_14      1          3         0    <none> (D)
  sll_19_14      1          3         0    <none> (D)
  sll_18_14      1          3         0    <none> (D)
  sll_17_14      1          3         0    <none> (D)
  sll_16_14      1          3         0    <none> (D)
  sll_15_14      1          3         0    <none> (D)

 (D) = wireload is default in technology library
rc:/>
```

Fig.14 Area Report of Floating Point Addition with Single Precision

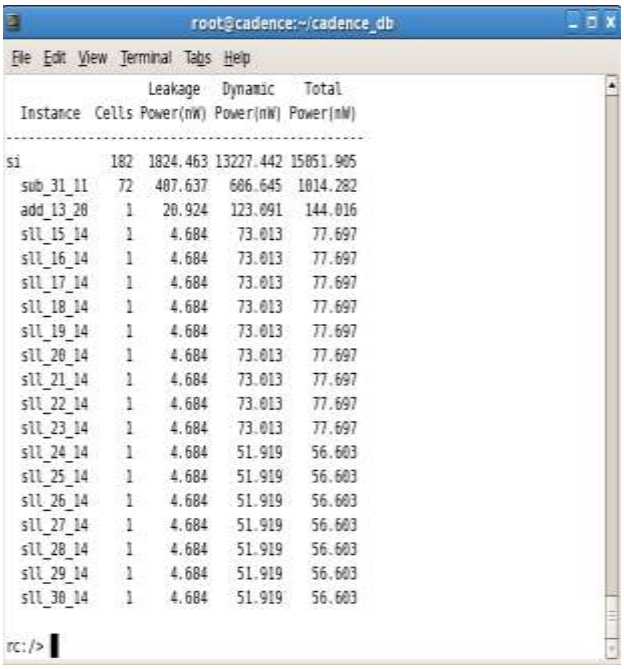


Fig.15. Delay Report of Floating Point Addition with Single Precision

Table 1: Results obtained for Floating Point Addition with Single Precision

Parameter	Proposed work
Area(m^2)	976
Leakage Power(nw)	1824.5
Dynamic Power(nw)	13227.4
Total Power(nw)	15051.905
Fan-out Load(fF)	885
Delay	230

3. CONCLUSION

In design of floating point addition with single precision several factors are taken into considerations like area, power and latency. The obtained results are effectively reducing the power consumption, area and latency with maximum delay of 327.6ns.The FP arithmetic typically requires additional operations such as alignment, normalization and rounding, giving rise to some significant increase in terms of area, power and computational latency. Reducing the complexities of FP calculations are carried out using single precision. But for demanding future applications the double precision and quadruple precision which will give accurate

REFERENCES

1. Bailey, D. (2013, April). High-precision computation: applications and challenges. In Proc. 21st IEEE Symp. Computer Arithmetic (ARITH-21), IEEE Press (p. 3).

2. Demmel, J., & Nguyen, H. D. (2013, April). Fast reproducible floating-point summation. In 2013 IEEE 21st Symposium on Computer Arithmetic (pp. 163-172). IEEE.

3. Kaivani, A., & Ko, S. (2015). Floating-point butterfly architecture based on binary signed-digit representation. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 24(3), 1208-1211

4. Qureshi, F., & Gustafsson, O. (2011, August). Generation of all radix-2 fast Fourier transform algorithms using binary trees. In 2011 20th European Conference on Circuit Theory and Design (ECCTD) (pp. 677-680). IEEE.

5. Swartzlander, E. E., & Lemonds, C. E. (Eds.). (2015). Computer Arithmetic: Volume III (Vol. 3). World Scientific

6. Tan, D., Lemonds, C. E., & Schulte, M. J. (2008). Low-power multiple-precision iterative floating-point multiplier with SIMD support. IEEE Transactions on Computers, 58(2), 175-187.

7. Jaiswal, M. K., & So, H. K. H. (2015, October). Dual-mode double precision/two-parallel single precision floating point multiplier architecture. In 2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC) (pp. 213-218). IEEE.

8. Manolopoulos, K., Reisis, D., & Chouliaras, V. A. (2016). An efficient multiple precision floating-point Multiply-Add Fused unit. Microelectronics Journal, 49, 10-18.

9. Kahan, W. (1996). IEEE standard 754 for binary floating-point arithmetic. Lecture Notes on the Status of IEEE, 754(94720- 1776), 11.

10. Cornea-Hasegan, M., & Norin, B. (1999). IA-64 floating- point operations and the IEEE standard for binary floating-point arithmetic.