Simulation and Synthesis Model for the Addition of Single Precision Floating Point Numbers Using VERILOG

Ravi Payal

School of Electronics, CDAC,Noida

1. Abstract

Single Precision floating point adder designed to fully conform to the IEEE 754 standard. In its fully-featured format, this adder supports all rounding modes: round to nearest even, round to plus infinity, round to minus infinity, and round to zero; it supports de-normalized numbers as both input and output; it supports all applicable exception flags: overflow, underflow, inexact, and invalid; it supports trapped overflow and underflow, where the result is returned with an additional bias applied to the exponent. The main objectives of the paper are the following is to design an adder for two positive floating point binary numbers which support all rounding modes.Therefore, Verilog programming for IEEE single precision floating point multiplier module have been explored.

Keywords - Floating point, Verilog, Precesion Synthesis, Xilinx

2. INTRODUCTION

Floating point describes a method of representing real numbers in a way that can support a wide range of values. Numbers are, in general, represented approximately to a fixed number of significant digits and scaled using an exponent. The base for the scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form:

$$\text{Significant digits} \times \text{base}^{\text{exponent}}$$

The student author designed the model for the Addition of two Positive Floating Numbers using Verilog. Verilog is a hardware description language (HDL) used to model electronic systems. This binary floating point adder designed to fully conform to the IEEE 754 standard. In its fully-featured format, this adder supports all rounding modes: round to nearest even, round to plus infinity, round to minus infinity, and round to zero; it supports denormalized numbers as both input and output; it supports all applicable exception flags: overflow, underflow, inexact, and invalid; it supports trapped overflow and underflow, where the result is returned with an additional bias applied to the exponent.

Due to performance constraints, only a subset of this standard is implemented in our paper. This standard gives an overview of floating point and its representation, including: floating-point numbers, special values such as zero, infinity, NaN, special operations and rounding modes.

2.1 IEEE Standard 754 Floating Point Numbers

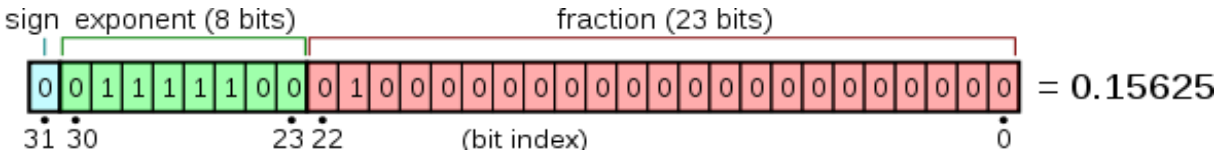
IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macintosh, and most UNIX platforms. Fixed point places a radix point somewhere in the middle of the digits, and is equivalent to using integers that represent portions of some unit. Another approach is to use rationals, and represent every number as the ratio of two integers.

Floating-point representation is the most common solution. It is basically represents real numbers in scientific notation. Scientific notation represents numbers as a base number and an exponent. Floating-point, on the other hand, employs a sort of "sliding window" of precision appropriate to the scale of the number. This allows it to represent numbers from 1,000,000,000,000 to 0.0000000000000001 with ease.

2.2 Single precision floating-point format

In IEEE 754-2008 the 32-bit base 2 format is officially referred to as binary32. It was called single precision in IEEE 754-1985. In older computers, some older floating point format of 4 bytes was used. This format is used due to its wider range over fixed point (of the same bit-width), even if at the cost of precision. The IEEE 754 standard specifies a binary32 as having:

- Sign bit (1 bit): Sign bit determines the sign of the number.
- Exponent width (8 bits): Exponent is either an 8 bit signed integer from -128 to 127 or an 8 bit unsigned integer from 0 to 255.
- Significance and precision: 23 bits



3. Algorithm for Adder Unit

The adder provides a partial implementation of IEEE 754 addition. The algorithm for floating point addition is actually particularly heinous to implement, involving several carefully-made bit selects, several shifts, and in its full form normalization and rounding using guard digits to determine how the final significance and must be rounded, if necessary. The full algorithm for addition in its simplest form is as follows:

a) Exponent subtraction: Perform subtraction of the exponents to form the absolute difference |E_a - E_b| = d.
b) Alignment: Right shift the significance and of the smaller operand by d bits. The larger exponent is denoted E_f.
c) Significance and addition: Perform addition on the operands.
d) Normalization: Normalize the significance and update E_f appropriately.
e) Rounding: Round the final result by conditionally adding one. If rounding causes an overflow, perform a 1-bit right shift and increment E_f.

The student author implemented required algorithm in Verilog HDL. First, the six sub blocks in Verilog were designed and after that these sub blocks were instantiated in a single Verilog Module. For the simulation result, the Test Bench was coded in Verilog HDL and the different test cases were defined.

3.1 Design

3.3.1 Inputs and Outputs

The adder takes three inputs: the two floating point operands and a control field (Fig. 1).

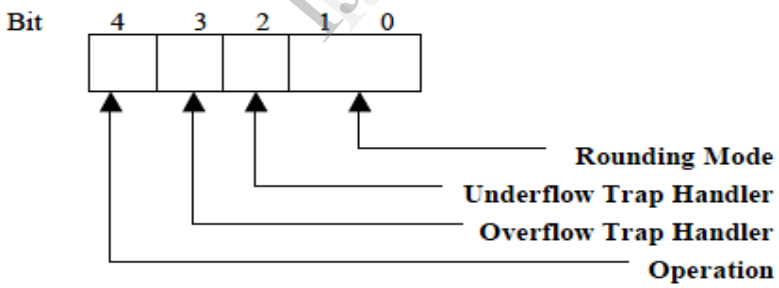


Fig. 1: The Control Field

The control field is 5 bits long. To enable the trap handlers, the appropriate bit is set high. The two least significant bits (LSBs) define the rounding mode as shown in Table 1. Bit 4 is used to select the operation, as shown in Table 2.

Control Field Bit		
1	0	Rounding Mode
0	0	Round to nearest even
0	1	Round to zero
1	0	Round to plus infinity
1	1	Round to minus infinity

Table 1: Description of rounding mode bits

Bit 4	Operation
0	Addition
1	Subtraction

Table 2: Operation Selection Bit

The adder has two outputs: the floating point result, and a flag field (Fig. 2). Appropriate flag bit will be set if respective condition is satisfied.

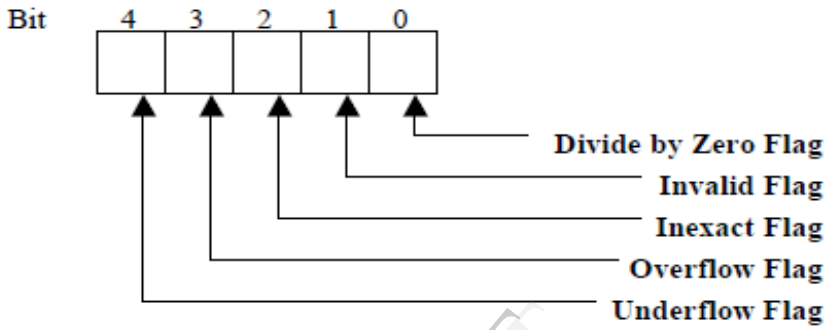


Fig. 2: The Flag Field

3.3.2 Sub Module Description

a) fpalign

The inputs are checked for zero which is determined by its exponent (if the exponent is all zeros) then the input is called zero. This means that denormalized inputs will result in a high zero input flag. So that denormalized numbers are handled correctly as its exponent part is zero. The inputs are then separated into the larger and smaller through a comparison of all their exponent and significand bits. The amount to shift the significand to align them properly is determined by the difference between the two exponents. If this shift amount is more than the number of bits in the significand plus three, the significand that is being shifted can be just shifted by the number of bits in the significant plus three. This will result in correct addition while still maintaining all sticky and guard bit information. The significant of the smaller input is right shifted by the calculated amount.

2) Mantadd

The shifted mantissas are then sent to the adder. Only the top [bits in significant + 3] bits of the mantissas are added, because this is the minimum number needed to get an accurate result. This means many of the bits of the smaller mantissa are not involved in the addition (since it was right shifted), and the most significant of these bits is stored as the guard bit, and the OR of the rest is stored as the pre-sticky bit.

3) Normalize

The result of the addition or subtraction is normalized. First, the number of leading zeros is determined with logic similar to a priority encoder. It is the amount to shift the mantissa for normalization. The exponent also has to be adjusted to make up for the shift, so the shift amount is also subtracted from the larger exponent. For denormal results, instead of left shifting by the amount of leading zeros, the sum is left shifted by the value of the bigger exponent. After all the shifting, the final significand is determined by considering the pending special cases. The rejected bits are ORed to get the sticky bit and the round bit is the XOR of the guard and sticky bits. Some useful flags are also determined at this point. If all the bits of the result of the effective operation on the mantissas are zero, the zero result flag is set. If either the round or sticky bits are high, the inex flag is set. If after addition overflows exits then when overflow trap handler is high then overflow flag is set.

4) Rounder

Now the final significand is rounded according to the respective rounding modes (whichever is set in control field). If the final significand is incremented, it could overflow. The final exponent has to be adjusted. MSB of the sum, actually the overflow bit. Since the final exponent has to be adjusted by one anyway, the case of overflow on rounding is accounted for by an incrementing of two rather than one. In binary, incrementing by 2 is almost the same as incrementing by one, except the incrementing starts at the second-to-least significant bit, rather than the LSB. At this point the final, correctly rounded, significand has been determined, as well as the final exponent

5) Final

Finally, the exception flags are determined and the final result is pieced together and placed into 'result'. Overflow is signalled when the exponent overflows or is all ones and overflow is trapped. Underflow is signalled when the result is tiny and inexact or if underflow is trapped, when the result is tiny. Invalid is signalled if an input was a NaN or infinity. Inexact is signalled if the result was inexact or if an untrapped overflow/underflow occurred, and the inputs weren't special cases.

6) Special

This is part of a floating point adder which sets special internal flags for floating point inputs : infinity, NaN, signaling NaN.

Representation of Special-Case Numbers in IEEE 754-1985 Standard

Meaning	Sign Field	Exponent Field	Mantissa Field
Zero	Don't care	All 0's	All 0's
Positive Denormalized	0	All 0's	Non-zero
Negative Denormalized	1	All 0's	Non-zero
Positive Infinity	0	All 1's	All 0's
Negative Infinity	1	All 1's	All 0's
Not a Number (NaN)	Don't care	All 1's	Non-zero

3. Result

The code for the Addition of two positive single precession floating point number was written in Verilog HDL. The different Sub blocks were implemented in Verilog HDL mostly using Data flow style of Modeling. After that all these blocks were instantiated in a single module. Test cases for Different combination were coded. The code was simulated in Mentor Graphics tool "ModelSim-6.5c". The simulation results were obtained in its wave form window. All the Required conditions are achieved. The output waveforms for different conditions are shown below.

3.1 Output waveform of two normalize inputs

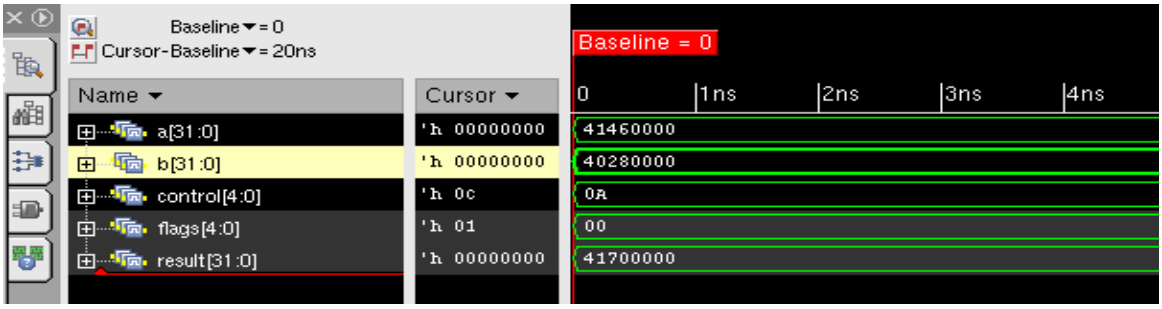


Fig3.When both the inputs are normal, the output is also normal.

3.2 Output Waveform when both the input are zero

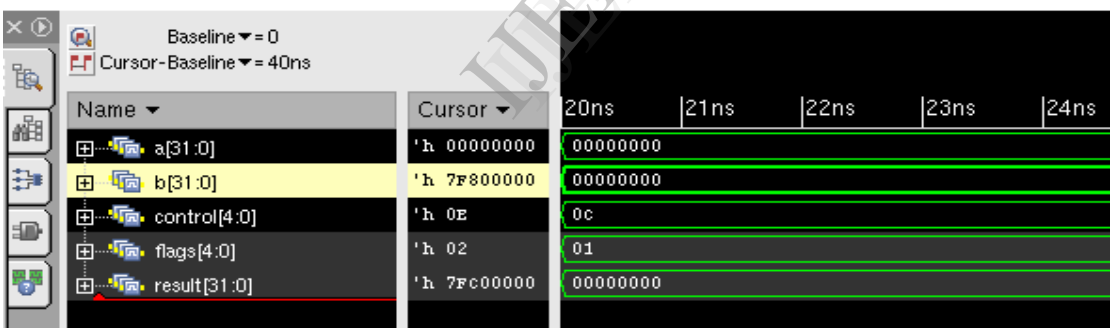


Fig4.When both the inputs are zero, the output is also zero.

3.3 Output waveform when one input is zero and other input is infinity

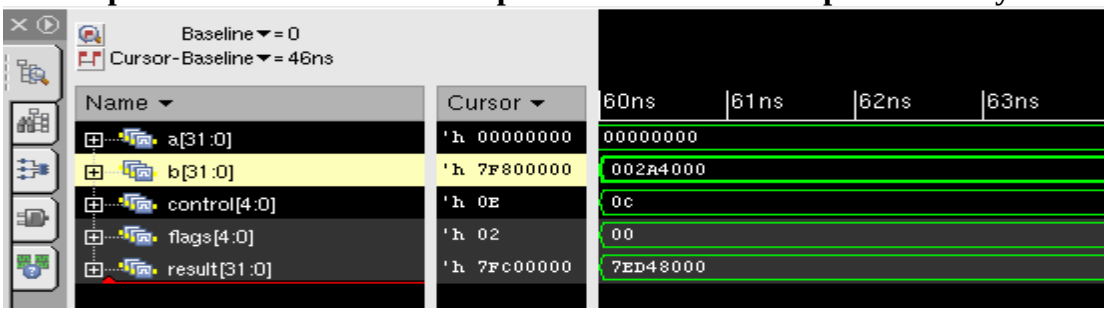


Fig5.When one input is zero and another input is infinity, the output is infinite.

3.4 Output waveform when one input is zero and other is de-normalize input

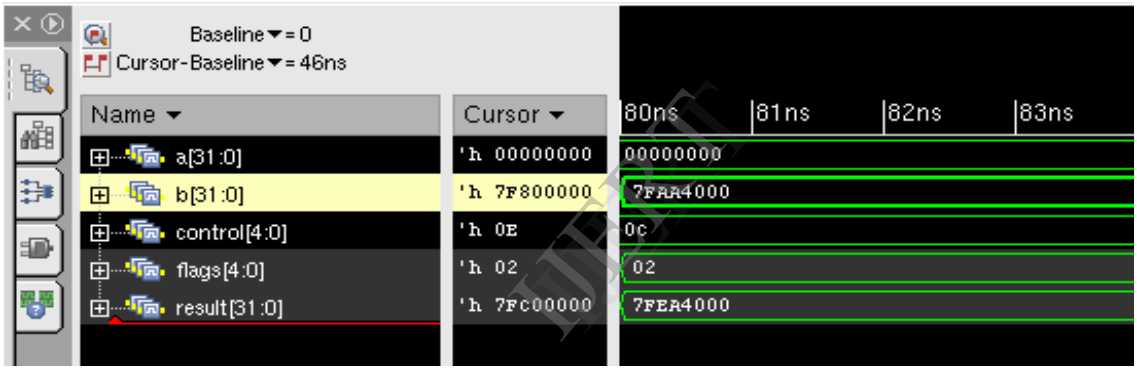


Fig6.When one input is zero and another input is denorm, the output is normal.

3.5 Output waveform when one input is zero and other is NaN input

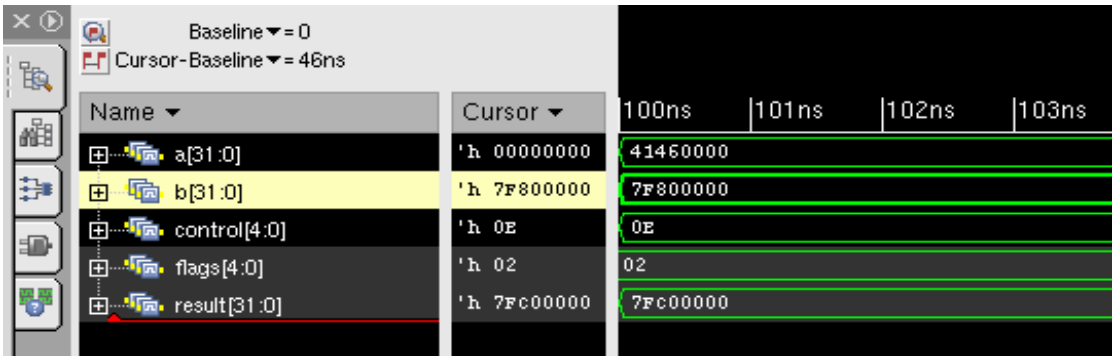


Fig7.When one input is zero and another input is NaN, the output is NaN.

a) Synthesis Result

The code was successfully synthesized using EDA tool "Precision RTL Plus". The target device was Xilinx - VIRTEX-6: 6VLX75TFF484. The number of LUTs slices utilized was 519, the number of CLBs used was 65 and the number of bonded IOBs used was 104. The Slack calculated was 1.033ns. The RTL schematic view, RTL hierarchy schematic and critical path are shown below as Fig. 8, Fig. 9 and Fig. 10 respectively.

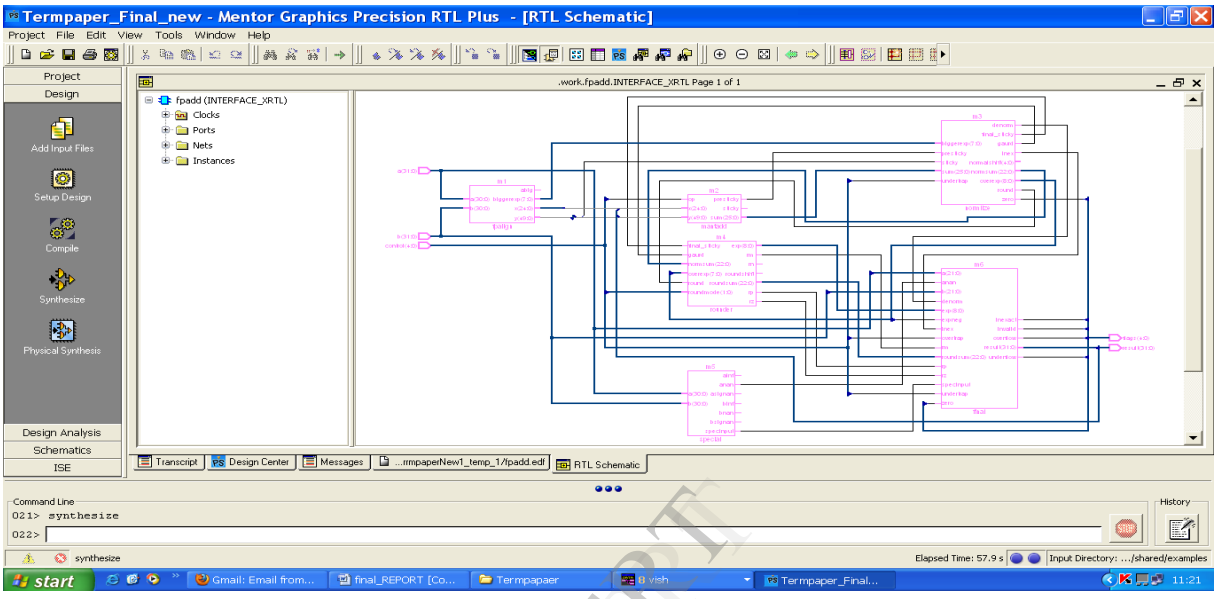


Fig 8: RTL Schematic View

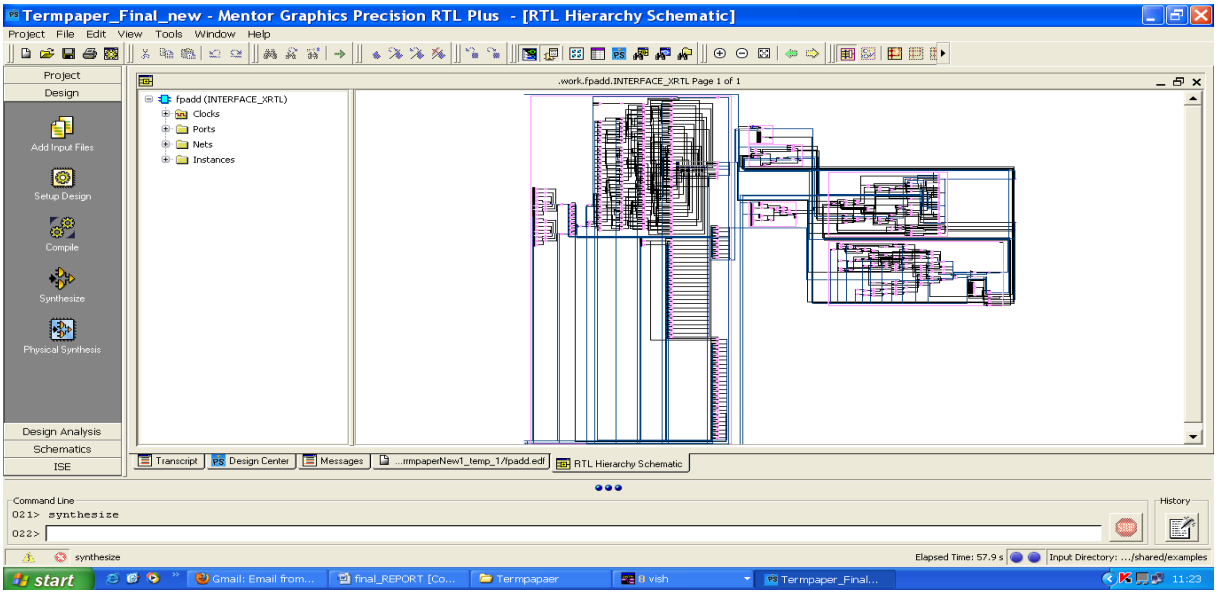


Fig 9: RTL Hierarchy Schematic View

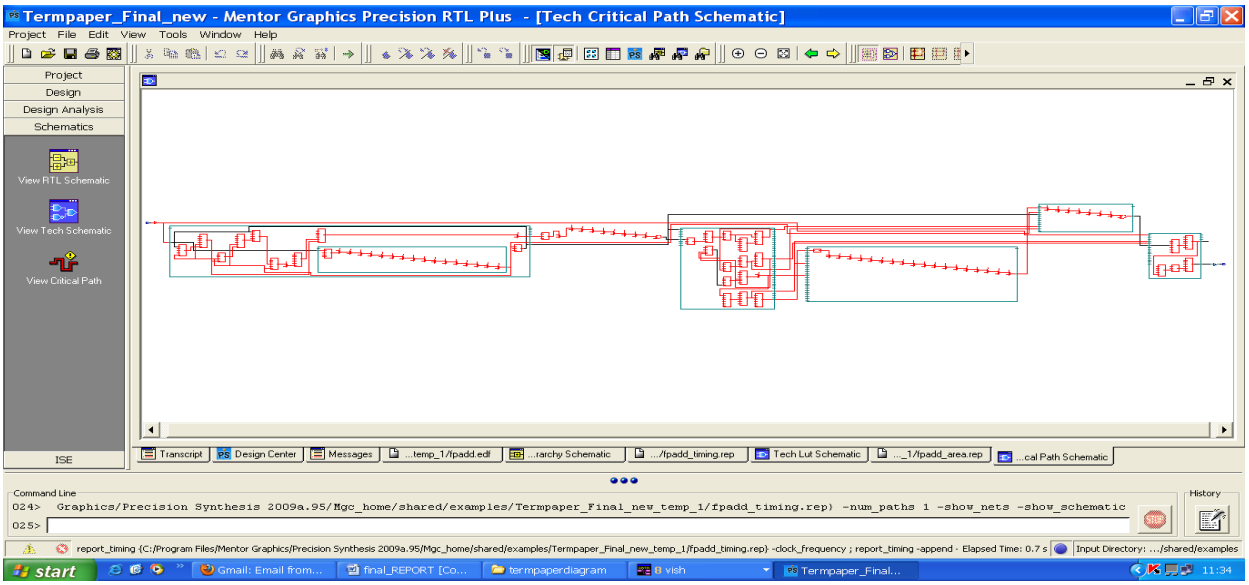


Fig. 10: Critical Path

4. Conclusion

The results obtained are very encouraging. The student author has successfully added many floating point numbers. As mentioned earlier, in this paper the student author has only taken single precision positive binary floating point numbers which conform to the IEEE 754 standards. The functional units are flexible and also give output for various special inputs like when zero, infinity value is given. It also supports demoralized numbers as both input and output. It also supports all applicable exception flags: overflow, underflow, inexact, and invalid; along with trapped overflow and underflow, where the result is returned with an additional bias applied to the exponent.

5. References

[1] L. Eisen, J.W.Ward III, H.-W. Tast, N. Mading, J. Leenstra,S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz, and S. R. Carlough. IBM POWER6 accelerators: VMX and DFU. IBM Journal of Research and Development, 51(6):663.684, 2007.

[2] A. Y. Duale, M. H. Decker, H.-G. Zipperer, M. Aharoni, and T. J. Bohizic. Decimal _oating-point in z9: An implementation and testing perspective. IBM Journal of Research and Development, 51(1/2):217.228, 2007

[3] Irvin Ortiz Flores," Optimizing the Implementation of Floating Point Units for FPGA Synthesis" Electrical and Computer Engineering Department

[4] Sukhdep kaur Bhatia,Ravi Payal, Dinesh chandra "A Fully Synthesizable Single Precision Floating Pony Multiplier"

[5] Verilog Hdl, Samir Palnitkar

[6] Wakerly, John F., "Digital Design – Principles and Practices", Tata McGraw Hill Series