Design and Simulation of Pipelined Floating-Point Multiplier using Logisim

Shivani Desai, Qasim Bhatia

Abstract: ***Floating point multiplication is a cycle intensive operation used in signal and image processing. The operation time of multiplication could be increased by pipelining this process. In this paper, we design and simulate a pipelined floating-point multiplier using the Logisim simulation tool. The numbers are stored in the IEEE 754 single-precision format. This circuit can be implemented into newer microprocessors to be used as a fast multiplier. An array of these multipliers can be used in matrix multiplications in artificial neural networks and other applications where rapid multiplication is required. The Logisim simulation tool is used as it is easy to use and has a simple interface with powerful abstraction features. It is used in major universities to teach and simulate computer architecture.***

Keywords **: *Floating-point multiplier, Pipeline Clock cycle***

I. INTRODUCTION

Logisim [1] is a digital circuit simulation tool used to teach introductory digital circuit design. It is a versatile tool and can support complex components and sub-circuits. The design of sub-circuits enables users to create modular circuits. These sub-circuit then can be bundled in libraries and can reused to build circuits higher complexity. This innovative use of abstraction makes Logisim a popular tool for teaching and learning in many top computer science institutions.Multiplication of any floating-point numbers can be done in three steps [4], first, the signs of the two numbers are checked to determine the sign of the product using standard rules of multiplication. Second, the exponents of the normalized floating-point numbers are added to get the exponent of the product. Lastly, the mantissas of the numbers are multiplied to get the mantissa of the product, this result is normalized and any exponent generated due to this is added to the exponent produced in the previous step. This procedure can be used to multiply two binary floating-point numbers.

The numbers are assumed to be in the standard IEEE 754 Single Precision format [6], which states that a 32-bit string is divided as follows:

- The most significant bit is designated as the sign bit.
- The 8 bits from 30-23 are store the exponent with a bias of 127(0111111).
- The remaining 23 bits are used to store the mantissa of thenormalized binary number, with the one before thedecimal point assumed to be included but not stored.

Another assumption taken is that the floating-point numbers to be multiplied are stored in an interleaved memory [10], so as to facilitate the reading of both numbers in a single clock cycle.

We then proceed to calculate the speed up of the pipeline by taking assumptions of the probable delay that will occur due to register transfers and propagation delays of each segment. This is done as Logisim cannot simulate delays in its execution environment.

II. PROCESS OF FLOAING POINT MULTIPLICATION

The binary numbers to be multiplied are fetched from memory. Their sign bits are then XORed to get the sign of the product. The exponents of the numbers are added and the mantissas (including the omitted one) are multiplied. If the product of the mantissas if greater than one, it is normalized by adding one to the exponent and shifting the binary left by one bit [5]. The result is combined to form the final product [11]. Shows the flowchart of the process.

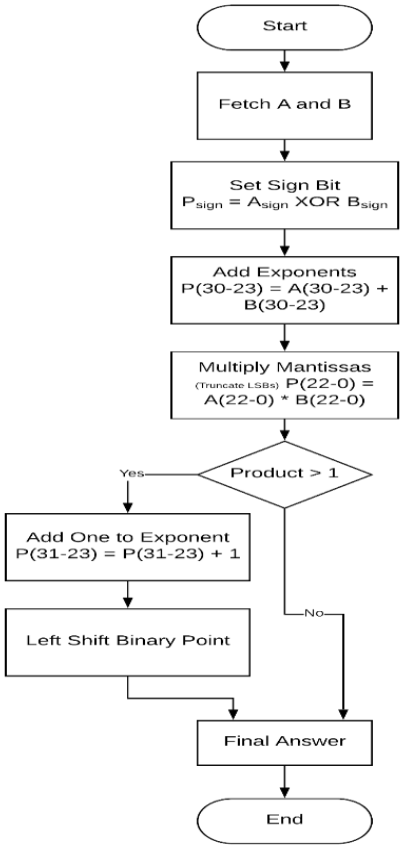

Fig. 1 Flowchart of Floating Point Multiplication [2]

Revised Manuscript Received on January 15, 2020

***** Correspondence Author
Shivani Desai*, CSE Department, Nirma University, Ahmedabad, India.
Qasim Bhatia, CSE Department, Nirma University, Ahmedabad, India.

III. DESIGN OF PIPELINED MULTIPLIER

The multiplication can be streamlined by dividing each process into three segments. The first segment computes the sign bit, the second adds the exponent, and the third multiplies and normalizes the mantissa.

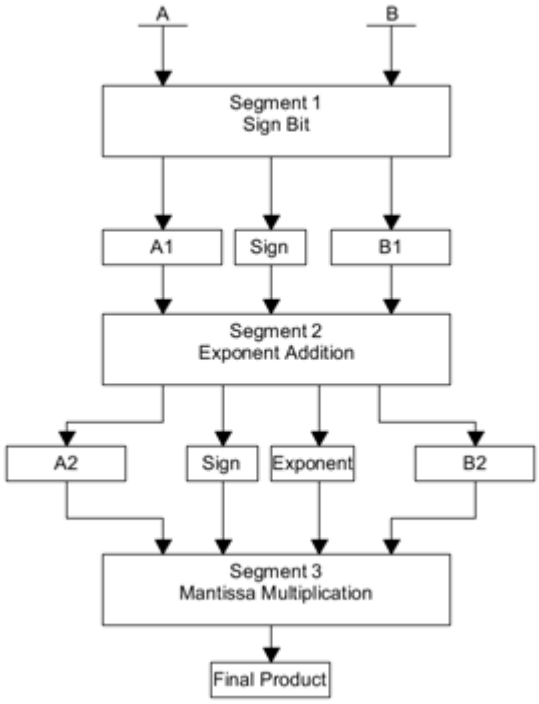

Fig. 2 Pipeline of Floating Point Multiplication [2]

The overview design of the pipeline is given in. The Space-Time diagram given in shows the movement of data through the pipeline, the figure shows four pairs of data. The diagram represents the utilization of the segments as a function of time [2].

Clock Cycles	1	2	3	4	5	6
1. XOR Sign	T1	T2	T3	T4		
2. Add Exponent		T1	T2	T3	T4	
3. Multiply MAntissa			T1	T2	T3	T4

Fig. 3 Space Time Diagram of Pipeline

IV. SIMULATION OF PIPELINE IN LOGISIM

The design mentioned above is simulated with by storing the numbers *A* and *B* in two separate ROMs so that they can be accessed simultaneously. Each segment is built in separate sub-circuits and are then combined in the main pipelined circuit.

A. Segment 1- Sign Bit Calculator

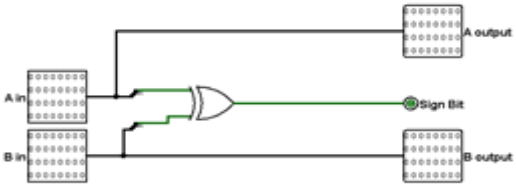

Fig. 4 Segment 1 of Pipeline – Sign Bit Generator [1]

shows the first segment of the pipeline. It receives the values of the two numbers from the memory and uses an XOR gate to set the sign bit of the product. The values and the generated sign bit are then propagated to the A1, B1 and Sign registers of the next segment.

B. Segment 2 – Exponent Addition

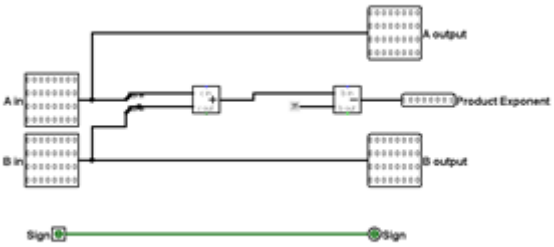

Fig. 5 Segment 2 of Pipeline – Exponent Adder [1]

shows the second segment of the pipeline. It uses the data of the registers A1 and B1 to calculate the exponent of the product. It does by taking bits 30-23 from both the registers with the use of a splitter and adds them together using the adder component available in Logisim. Since the sum of these components will be greater than the accepted range of the IEEE format, the bias of 127 (01111111) is subtracted from the sum to get the actual exponent of the product. The values A1, B1, Exponent and Sign are transfers to the registers of the next segment.

C. Segment 3 – Mantissa Multiplier

The circuit in depicts the final segment of the pipeline. It takes the bits 22-0 from A and B, adds one as the twenty-forth bit to recreate the mantissa. These 24-bit numbers are fed to the multiplier component. The numbers are multiplied to give a forty-eight-bit number with the binary point after the 47^{th} bit. The forty-eighth bit is checked, if it is one, then one is added to the exponent to normalize the mantissa and bits 46-24 of the product is placed in the answer's mantissa part (22-0). If it is not the case, then the bits 45-23 are chosen to be the part of the answer. This choice is made using a multiplexer with the select input being the 48^{th} bit.

D. Assembling the Pipeline

The pipeline is built by as the main circuit in Logisim, while the segments are included as sub-circuits. The main circuit includes the two ROMs for fetching the operands, the operands from the ROMs are fed to the first segment, one RAM to store the final product, four 32-bit registers for the intermediary outputs,

two one-bit flip-flops for the inter-segment sign bits and one eight-bit register for the exponent generated in segment 3. One counter is taken to be used as the address register for the memory units and to track the number of completed clock cycles. These components are connected as per with a clock to complete the pipeline.

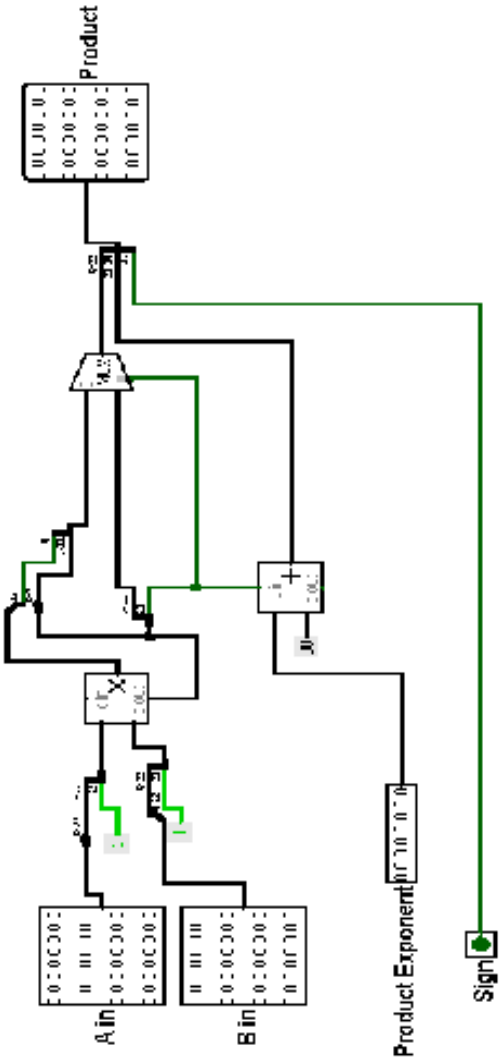

Fig. 6 Segment 3 of Pipeline - Mantissa Multiplier [1]

V. SIMULATION OF CIRCUIT

The circuit is tested by taking a few sample values of A and B and running them through the pipeline. The sample test cases are given in Table 1 Sample Test Cases in Decimal Table 1 and
Table 2. Values of A are stored in the first ROM while those of B are stored in the second ROM. Initially the pipeline is empty (shown in Fig. 7 Floating point multiplication pipeline) and requires three clock cycles to fill the pipeline. The RAM starts writing the output products from the third clock cycle. The following sections show the status of the pipeline during the first the three clock cycles and at the end of the sixth cycle, when all the inputs have been successfully calculated through the pipeline.

Table 1 Sample Test Cases in Decimal

	A	B	A * B
Set 1	125.125	12.0625	1509.3203
Set 2	125.25	3152	394788
Set 3	-12672	-660.25	8366688
Set 4	1.37109	-1.1406	-1.56390

Table 2 Sample Test Cases in IEEE 754 Format (Hex)

	A	B	A * B
Set 1	42FA4000	41410000	44BCAA40
Set 2	42FA8000	45450000	48C0C480
Set 3	C6460000	C4251000	4AFF54C0
Set 4	3FAF8000	BF920000	BFC82E00

A. Clock Cycle 1

At the end of the first clock cycle, the values of Set 1 (Table **2**) are in the A1, B1 and Sign1 registers while the values of Set 2 are being processed in Segment 1.

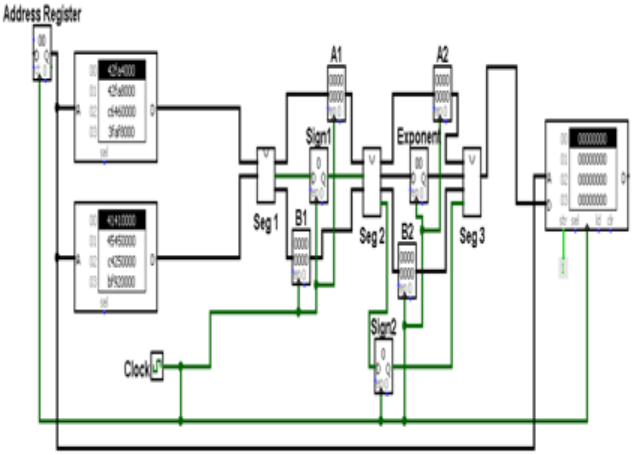

Fig. 7 Floating point multiplication pipeline [1]

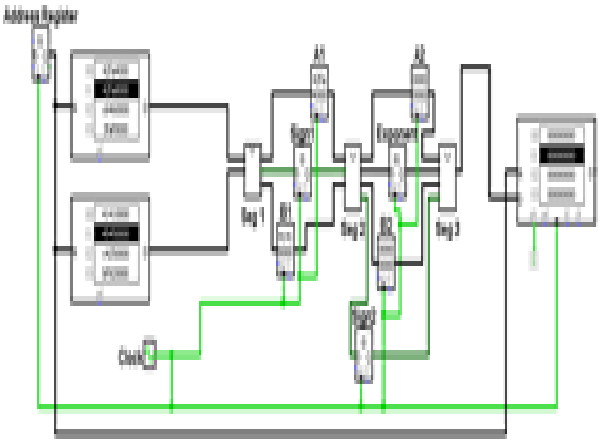

Fig. 8 Pipeline at the end of clock cycle 1 [1]

B. Clock Cycle 2

At the end of clock cycle 2, Set 3 is in segment 1, Set 2 is in the first intermediary registers and being processed in segment 2, Set 1 is in the second intermediary registers and being processed in segment 1.

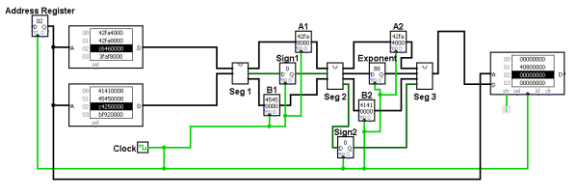

Fig. 9 Pipeline at the end of clock cycle 2 [1]

C. Clock Cycle 3

Finally, the output of Set 1 is stored in the RAM as clock cycle 3 completes, this also fills the pipeline and subsequent cycles compute the answer of the following sets and write them to memory.

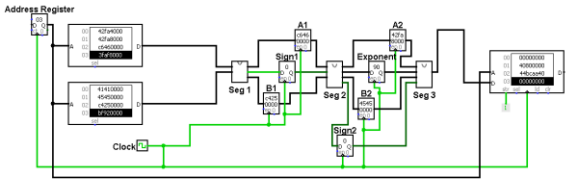

Fig. 10 Filled Pipeline at the end of clock cycle 3 [1]

D. Completion of all Four Tasks

The pipeline becomes empty and at the sixth clock cycle, it is seen in Fig. that four results are stored in RAM and the pipeline has been emptied.

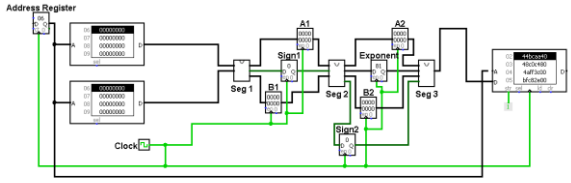

Fig. 11 All tasks completed, Empty Pipeline [1]

VI. COMPARISION TO A NON-PIPLINED MULTIPLIER

The speed up of typical pipeline when the number of tasks is large is given by

$$S = \frac{t_n}{t_p}$$

[2,7]

The three segments can give a theoretical speed-up of three, but this is not practical due the varying delays in each segment. Another drawback we see is that Logisim [1] cannot simulate propagation delays.

VII. PROBLEMS IN FLOATING POINT MULTIPLICATION

There is an inherent error in floating point multiplication. The value stored in binary is always a bit off from its actual values and hence multiplication of these flawed representations leads to compounding of this floating-point error [9]. This error is often in the range of 10^{-16} to 10^{-8} and can be ignored. If this error is not acceptable, then the numbers can be represented using the IEEE 754 double precision format [6]. The test for zero and infinity as input has also not been tested thoroughly [12].

VIII. CONCLUSION

Successfully designed and simulated a pipelined floating multiplier and studied its working using a few sample test-cases. Although this design may not represent the real-world scenario, by working on it we get a detailed view inside the working of pipelines and their ability to streamline the same task over different data. The design may be used as a framework to pipeline other cycle intensive arithmetic operations. Future work may involve simulation of the circuit on real hardware and even pipelining the other remaining operations of an arithmetic and logic unit of a microprocessor.

REFERENCES

1. Burch, Carl. "Logisim: a graphical system for logic circuit design and simulation." Journal on Educational Resources in Computing (JERIC) 2, no. 1 (2002): 5-16.
2. Mano, M. Morris. "Computer System Architecture, 1982." Englewood Cliffs, NJ,: 53-54.
3. Devi M, Vidya, Chandraprabha R, Mamatha K R, Shashikala J, and Seema Singh. "Design and Implementation of Pipelined Reversible Floating-Point Multiplier Using Carry Save Adder." International Journal of Recent Advances in Engineering & Technology (IJRAET) 2, no. 1 (January 2014): 14-17.
4. Floating Point Tutorial | IEEE 754 Floating Point Basics | Tutorials. Accessed April 17, 2018.
5. Al-Ashrafy, Mohamed, Ashraf Salem, and Wagdy Anis. "An efficient implementation of floating point multiplier." In Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International, pp. 1-5. IEEE, 2011.
6. Kahan, William. "IEEE standard 754 for binary floating-point arithmetic." Lecture Notes on the Status of IEEE 754, no. 94720-1776 (1996): 11.
7. Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.
8. Govindu, Gokul, et al. "Analysis of high-performance floating-point arithmetic on FPGAs." Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International. IEEE, 2004.
9. Muller, Jean-Michel, et al. Handbook of floating-point arithmetic. Springer Science & Business Media, 2009.
10. Rau, B. Ramakrishna. "Pseudo-randomly interleaved memory." ACM SIGARCH computer architecture news. Vol. 19. No. 3. ACM, 1991.
11. Yu, Robert K., and Gregory B. Zyner. "167 MHz radix-4 floating point multiplier." Computer Arithmetic, 1995., Proceedings of the 12th Symposium on. IEEE, 1995.
12. Hickmann, Brian, et al. "A parallel IEEE P754 decimal floating-point multiplier." Computer Design, 2007. ICCD 2007. 25th International Conference on. IEEE, 2007.
13. Underwood, Keith. "FPGAs vs. CPUs: trends in peak floating-point performance." Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays. ACM, 2004.

AUTHORS PROFILE

Shivani Desai pursed MTech and currently working as an assistant professor in CSE Department, Institute of Technology, Nirma University, Ahmedabad. Her main area of research work focuses on Digital Systems, Machine Learning and Internet of Things. She has published many papers in various area of her research.

Qasim Bhatia is a BTech student in the Department CSE, Nirma University. He has minored in Information Security.He was the lead designer in the NUtech'19 Participant Database Management System for the institute's annual technical festival. He has completed 3 projects in the areas of object-oriented programming, machine learning and blockchain. His research areas are in machine learning, blockchain and web security.