SEMARAK ILMU PUBLISHING

Semarak Engineering Journal

Journal homepage:
https://semarakilmu.com/
ISSN: 3036-0145



Register Transfer Level Design of 32-Bit RISC-V Out-of-Order Processor

Jia Yang Hor[1], Chia Yee Ooi[1,*], Soon Chieh Lim[2]

[1] Embedded System iKohza, Electronic Systems Engineering Department, Malaysia-Japan International Institute of Technology, Universiti Teknologi Malaysia, Jalan Sultan Yahya Petra, Kampung Datuk Keramat, 54100 Kuala Lumpur, Wilayah Persekutuan Kuala Lumpur, Malaysia
[2] SkyeChip Sdn Bhd, 1-18-12, Suntech @ Penang Cybercity, Lintang Mayang Pasir 3, Bandar Bayan Baru, 11950 Bayan Lepas, Penang, Malaysia

ARTICLE INFO

Article history:
Received 2 June 2023
Received in revised form 14 July 2023
Accepted 5 August 2023
Available online 3 September 2023

Keywords:
Out-of-order processor; reservation stations; Tomasulo algorithm

ABSTRACT

This work demonstrates the performance boost brought by a RISC-V out-of-order processor. In this work, we design a 32-bit RISC-V out-of-order processor using RTL methodology, unlocking the potential of out-of-order execution for enhanced performance in RISC-V processors. By comparing the out-of-order and in-order processors, this work highlights the limitations of the latter. The work focuses on Verilog code and the Synopsys VCS compiler [15] for designing both processors, with observation facilitated by the DVE waveform viewer. The RTL design process includes load buffers, reservation stations for adders and multipliers, and effective hazard and dependency handling through stall mechanisms. Incorporating the Tomasulo algorithm enables dynamic instruction scheduling in the out-of-order processor. This work's outcome is an RTL design of a 32-bit RISC-V out-of-order processor, demonstrating improved performance compared to the in-order processor [1]. This work sheds light on the advantages of out-of-order execution and paves the way for further research in advanced RISC-V processors. The results will showcase the significant advantage of the RISC-V out-of-order processor over the in-order processor, offering a glimpse into the exciting possibilities that out-of-order execution brings to the RISC-V architecture, leaving readers eager to delve deeper into the potential impact of this innovation.

1. Introduction

In-order processors are slower as compared to out-of-order processors because they are constrainedby the requirement to execute instructions in the order in which they are fetched from instruction memory. This means that the processor must wait for the previous instruction to complete before executing the next one. This can lead to a slowdown in performance [14].

This work draws inspiration from the Berkeley Out-of-Order Machine (BOOM) [2,3] and other related studies [5-8,13], which have demonstrated the advantages of out-of-order execution in improving processor performance. By implementing out-of-order execution, the work aims to overcome the sequential execution constraint of in-order processors, allowing instructions to be dynamically reordered for optimal execution. Additionally, by leveraging the open-source nature of

* *Corresponding author.*
E-mail address: ooichiayee@utm.my

RISC-V, the design benefits from the flexibility, modularity, and scalability offered by this instruction set architecture [9].

We design a 32-bit RISC-V Out-of-Order processor using Verilog. It aims to demonstrate the performance improvement of the RISC-V architecture with out-of-order execution compared to an in-order processor. The design will be tested using a sample instruction set to validate its functionality and performance.

The paper outline is as follows. Section 2 describes the proposed out-or-order RISC-V. Section 3 discusses the performance of the RISC-V and Section 4 concludes the works.

2. Proposed Out of Order RISC-V

This proposed design only focuses on integer operations, with the design and implementation of both RISC-V out-of-order and in-order processors at the Register Transfer Level (RTL) using Verilog. The design process begins with the classic 5-stage 32-bit RISC-V architecture of an in-order processor followed bythe 5-stage 32-bit RISC-V Out-of-order processor. Later, the performance and results of this in-order processor will be compared to the out-of-order processor. The 5-stage 32-bit RISC-V in-order processor, as depicted in Figure 1, consists of 5 stages which are Instruction Fetch, Instruction Decode, Execution, Memory, and Write Back stages which are also presented in 5-stage 32-bit RISC-V out-of-order processor as depicted in Figure 2. Note that forwarding paths [10] are also included in both processors.

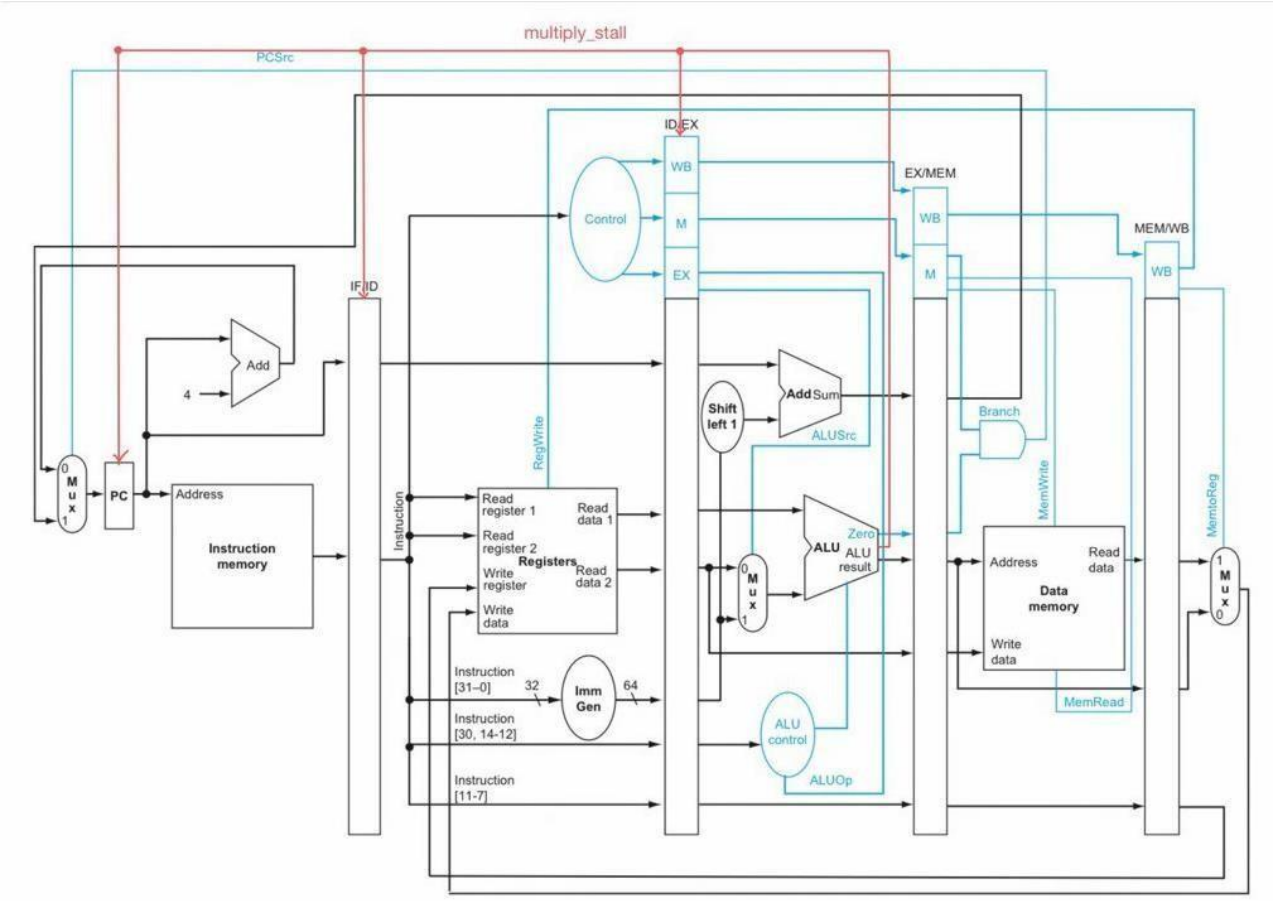


Fig. 1. The 5-stage 32-bit RISC-V architecture of in-order processor [4]

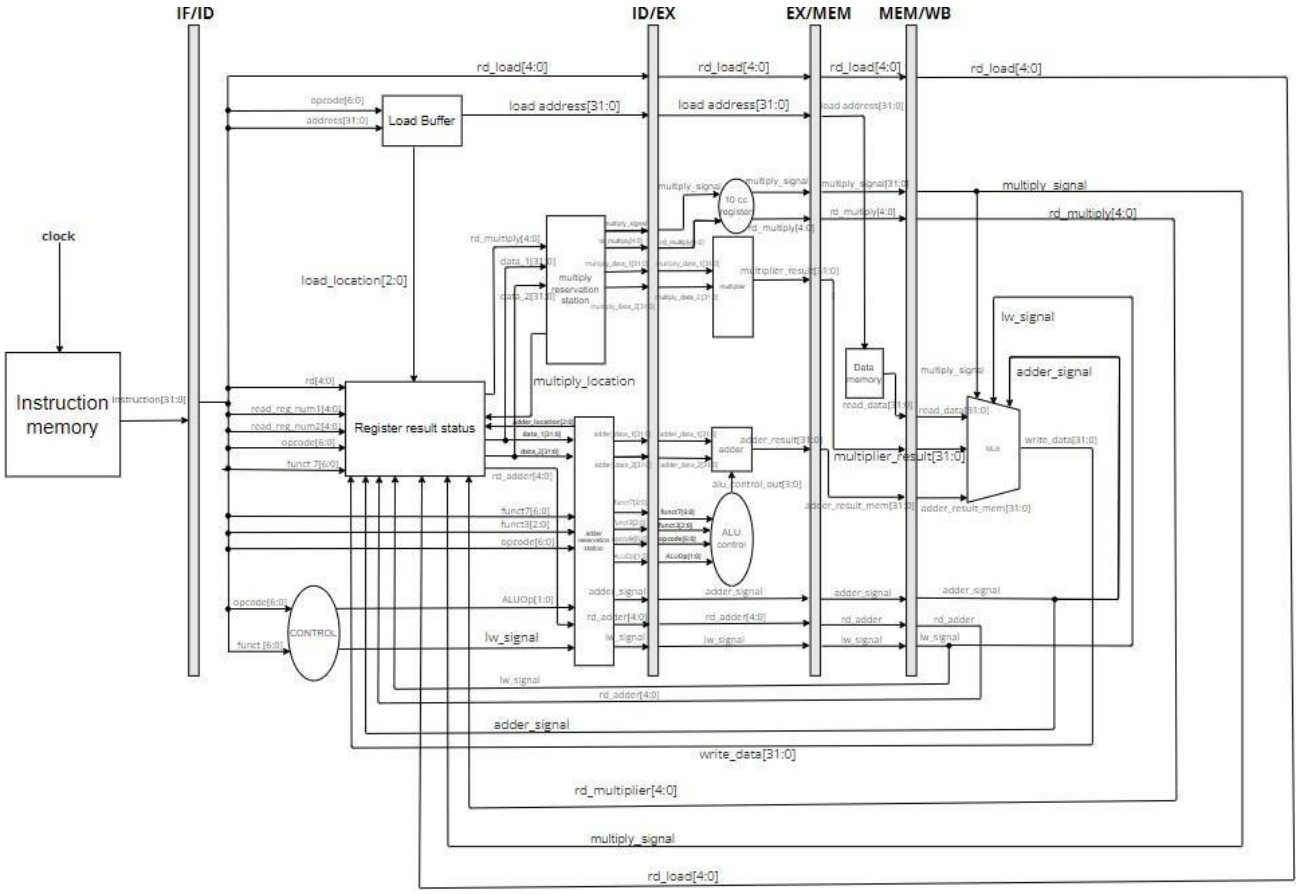


Fig. 2. The architecture of the 5-stage 32-bit RISC-V Out-of-order processor

The design of the 5-stage RISC-V Out-Of-Order processor in this work includes key components inspired by the Tomasulo algorithm [11] which are the reservation stations [12], load buffer and register result status. By integrating these elements into the processor's architecture, the design harnesses the advantages of dynamic instruction scheduling and out-of-order execution.

By referring to Table 1, a register in the register file holds a pointer or the actual result that should bewritten on it. If a register holds a pointer pointing to a Reservation Station (RS) entry or Load Buffer, this indicates that the register is a destination register and waiting for the result from the operation in that RS entry or load buffer. Otherwise, the register will be holding the actual result that is produced in the processor. In Table 1, “result_status” refers to the status of the register (busy or free) and “store_rd_id” refers to the pointer pointing to the reservation stations.

Table 1
The table of registered result status

Register	X0	…	X31
pointer/result			
result_status			
store_rd_id			

Within the context of the Tomasulo Algorithm, the reservation station as shown in Table 2 is a key component in out-of-order execution processors that manages and facilitates the execution of instructions. It acts as a centralized buffer or queue that holds instructions waiting for their operands to become available. There are 2 reservation stations in the design: adder reservation station which operates for ALU operation and multiply reservation station which operates for multiply operation.

There are several components in this reservation module which are Vj, Vk and Qj, Qk. Vj and Vk are the values of the source operands after they have been resolved whereas Qj and Qk are the pending operands in terms of RS identity. The "j" in Vj refers to the first register while the "k" refers to the second register; the same goes for Qj and Qk. The output of the reservation station should always dependon Vj and Vk. When both Vj and Vk have values, the corresponding instruction is ready for execution, so the values of Vj and Vk should be sent to the output of the reservation station.

Table 2
The reservation station

Location	Unit	Busy/free	Vj	Vk	Qj	Qk
4	Multiplier reservation 1	Free				
5	Multiplier reservation 2	Free				
6	Adder reservation 1	Free				
7	Adder reservation 2	Free				
8	Adder reservation 3	Free				

3. Result

Figure 3 presents a list of basic RISC-V instructions that are used to test the RISCV-V processors. lw instruction takes 3 cycles, mul takes 13 cycles, and sub takes 5 cycles. Multiplication operation takes more cycles due to the inherent complexity of the multiplication operation, which requires more timecompared to simpler operations.

```
lw      x6, 34(x2)
lw      x2, 45(x3)
mul     x0, x2, x4
sub     x8, x6, x3
sub     x10, x6, x2
```

Fig. 3. Instruction set used for testing the RISC-V processors

Figure 4 shows the output waveform that is produced by the RISC-V in-order processor. The start timeof this in-order processor is 120s while its end time is 630s. The reason that it is starting from 120s instead of 0 is because according to the design of the in-order processor in this work, the first 120 seconds are used to store the value to the source registers of load instruction in Figure 3. The value must be stored into the register before loading to them. Otherwise, there will be no value loading tox6 and x2. CPU time and speedup are two common evaluation metrics for processors [16].

CPU time (or CPU Execution time) is the time between the start and the end of execution of a given program. There are two ways to calculate the CPU time which are stated in Eq. (1) and (2)

$$\text{CPU time} = \text{Clock cycles for a program} * \text{Clock cycle time} \quad (1)$$

$$\text{CPU time} = \text{Time at which it completes all instructions} - \text{Time where it begins} \quad (2)$$

The total number of clock cycles in Figure 4 to complete all the instructions is 25.5 and each clock cycleis 20ns. Applying Eq. (1), the CPU time of the in-order processor is calculated by multiplying 25.5 by 20ns, resulting in 510ns. Similarly, applying Eq. (2), the CPU time is calculated by subtracting120ns from 630ns, also resulting 510ns. Consequently, the CPU time of the RISC-V in-order processoris determined to be 510ns.



Fig. 4. Output waveform of RISC-V In-order processor

Figure 5 shows the output waveform produced by the out-of-order processor. In this case, the startingpoint is 0s. This is because, for the design of an out-of-order processor, the source registers of the lw instructions are initialized with values beforehand. Therefore, the time is measured when instructionis fetched in the beginning.

The total number of clock cycles in Figure 5 to complete all the instructions is 20.5 and the clock cycleis the same as previous. Applying Eq. (1), the CPU time of the in-order processor is calculated by multiplying 20.5 by 20ns, resulting in 410ns. Similarly, applying Eq. (2), the CPU time is calculatedby subtracting 0ns from 410ns, also resulting in 410ns. As a result, the CPU time of the RISC-V out-of-order processor is 410 ns.

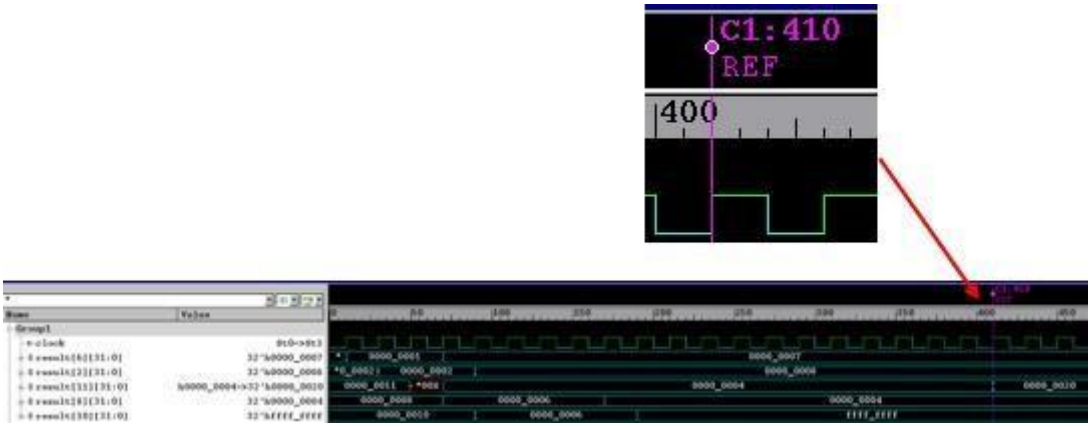


Fig. 5. Output waveform of RISC-V Out-of-Order processor

The difference in CPU time between the in-order processor and out-of-order processor is as shown in Eq. (3) while the Speedup is as shown in Eq. (4)

CPU time difference =
$$(\text{CPU time of in} - \text{order processor}) - (\text{CPU time of out} - \text{of} - \text{order processor}) \quad (3)$$

$$Speedup = \frac{\text{Execution time of in} - \text{order processor}}{\text{Execution time of out} - \text{of} - \text{order processor}} \quad (4)$$

Therefore, calculating the difference in CPU time between the in-order processor and out-of-order processor using Eq. (3) yields 510ns - 410ns = 100ns, while the calculated speedup is 510ns / 410ns = 1.243. As a result, the RISC-V out-of-order processor is 100 seconds and 1.243 times faster than the RISC-V in-order processor.

Figure 5 reveals non-sequential changes in register values compared to the instructions. For example,lw instruction writes to x6 in the 4th clock cycle, while x2 is written in the subsequent 5th cycle. The value of x11, resulting from the mul instruction, requires 14 clock cycles to obtain the final result. However, due to data dependencies, an additional 4 clock cycles are required to complete the computation, making it complete in 18 clock cycles. While waiting for x11 to obtain its result, bothx8 and x10, originating from the sub instruction, finish their computations earlier. This highlights the impact of data dependencies and the varying completion times of instructions within the execution pipeline. Table 3 presents the results that indicate the speedup increased significantly as the number of instructions grew from 5 to 10. The calculated speedup for 5 instructions was 1.243, while for 10 instructions it reached 1.488. These findings demonstrate a notable improvement in speedup when both the in-order and out-of-order processors were tested with a larger instruction set.

Table 3
Relation between the speedup and the number of instructions

Number of instructions	Execution time of in-order processor(ns)	Execution time of out-of-order processor(ns)	speedup
5	510	410	1.243
6	530	410	1.293
7	550	410	1.341
8	570	410	1.390
9	590	410	1.439
10	610	410	1.488

The RISC-V out-of-order processor in this work showed significant superiority over the in-order processor. With the tested instruction set, it achieved a speedup of 1.243. When evaluated with a larger instruction set, the speedup increased further to 1.488. These results highlight the improved performance and advantage of the designed RISC-V out-of-order processor, especially with larger instruction sets. By dynamically scheduling instructions and utilizing parallelism efficiently, the out-of-order processor enhances its performance.

4. Conclusion

In conclusion, the evaluation and comparison between the out-of-order and in-order processors revealed significant findings. The out-of-order processor exhibited superior performance in terms of execution time delivering faster processing speeds. This can be attributed to its advanced instruction scheduling and dynamic execution capabilities, enabling efficient resource utilization and effective parallelization of instructions. The out-of-order processor prioritized instructions based on data availability, facilitating the early completion of dependent instructions. The experiment has demonstrated that out-of-order execution performs better in terms of execution time and speedup. These results affirm the advantages of employing out-of-order execution techniques, optimizing resource utilization, minimizing data dependencies, and achieving faster processing times. This is beneficial in complex and data-intensive applications where instruction-level parallelism and efficient resource management are crucial.

5. Acknowledgement
The author also extends thanks to the faculty of Malaysia-Japan International Institute of Technology Universiti Teknologi Malaysia for providing the necessary resources, facilities, and opportunities to carry out this work.

References

[1] Le, Anh-Tien, Trong-Thuc Hoang, Ba-Anh Dao, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham. "A cross-process spectre attack via cache on RISC-V processor with trusted execution environment." *Computers and Electrical Engineering* 105 (2023): 108546. https://doi.org/10.1016/j.compeleceng.2022.108546

[2] "The Berkeley Out-of-Order Machine (BOOM)." RISCV-BOOM, docs.boom- core.org/en/latest/sections/intro-overview/boom.html#the-berkeley-out-of-order-machine- boom.

[3] Celio, Christopher, David A. Patterson, and Krste Asanovic. "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor." *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167* (2015).

[4] Patterson, David A., and John L. Hennessy. *Computer organization and Design*. Morgan Kaufmann, 1994.

[5] Gao, Jie, and Jun Zhang. "Research and Design of RISC-V Four-Stage Out-of-Order Execution Processor." In *2022 IEEE 16th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*, pp. 1-3. IEEE, 2022. https://doi.org/10.1109/ICSICT55466.2022.9963329

[6] Mashimo, Susumu, Akifumi Fujita, Reoma Matsuo, Seiya Akaki, Akifumi Fukuda, Toru Koizumi, Junichiro Kadomoto et al. "An open source FPGA-optimized out-of-order RISC-V soft processor." In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pp. 63-71. IEEE, 2019. https://doi.org/10.1109/ICFPT47387.2019.00016

[7] Martina, Maurizio, and Marco Andorno. "Design of the frontend for LEN5, a RISC-V Out-of-Order processor."

[8] Sarihi, Amin, Michael A. Schoenfelder, and Abdel-Hameed A. Badawy. "Performance Evaluation of an Out-of-Order RISC-V CPU: A SPEC INT 2017 Study." In *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pp. 418-419. IEEE, 2022. https://doi.org/10.1109/IPCCC55026.2022.9894297

[9] "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20190608-Base-Ratified", Editors Andrew Waterman and Krste Asanovi´c, RISC-V Foundation, March 2019

[10] Eren, Ilknur. "Five Stages of RISC Pipeline." Medium, 28 Jan. 2021, levelup.gitconnected.com/five-stages-of-risc-pipeline-aad0c3eb1233.

[11] Lee, Jongbok. "The Design and Simulation of Out-of-Order Execution Processor using Tomasulo Algorithm." *The Journal of the Institute of Internet, Broadcasting and Communication* 20, no. 4 (2020): 135-141.

[12] Spasov, Dejan. "A Circuit for Identifying Oldest Ready Instructions in Reservation Stations." In *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pp. 109-113. IEEE, 2020. https://doi.org/10.23919/MIPRO48935.2020.9245125

[13] Miyazaki, Hiromu, Takuto Kanamori, Md Ashraful Islam, and Kenji Kise. "RVCoreP: An optimized RISC-V soft processor of five-stage pipelining." *IEICE TRANSACTIONS on Information and Systems* 103, no. 12 (2020): 2494-2503. https://doi.org/10.1587/transinf.2020PAP0015

[14] Mallidu, Jayashree, and Saroja V. Siddamal. "A Survey on In-Order 5-Stage Pipeline RISC-V Processor Implementation." In *Information and Communication Technology for Competitive Strategies (ICTCS 2021) Intelligent Strategies for ICT*, pp. 777-784. Singapore: Springer Nature Singapore, 2022. https://doi.org/10.1007/978-981-19-0098-3_73

[15] Deshpande, Neha, and K. B. Sowmya. "A review on asic flow employing eda tools by synopsys." *SSRG International Journal of VLSI & Signal Processing (SSRG-IJVSP)* 7, no. 1 (2020). https://doi.org/10.14445/23942584/IJVSP-V7I1P104

[16] Stallings, William. *Computer organization and architecture: designing for performance*. Pearson Education India, 2003.